Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miloš Lukić

Evalvacija procesov razvoja programske opreme s pomočjo dnevniških zapisov razvojnega okolja

magistrsko delo

magistrski program druge stopnje
računalništvo in informatika

Mentor: doc. dr. Damjan Vavpotič
Somentor: doc. dr. Tomaž Hovelja

Ljubljana, 2017

Avtorske pravice. Rezultati magistrskega dela so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov magistrskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

©2017 Miloš Lukić

Zahvala

Zahvaljujem se mentorju doc. dr. Damjanu Vavpotiču in somentorju doc. dr. Tomažu Hovelji za pomoč in usmerjanje ob nastajanju tega magistrskega dela. Zahvaljujem se tudi zaposlenim v vseh treh podjetjih za aktivno sodelovanje v raziskavi. Zahvaljujem se Dariji za napotke pri pisanju in Urošu za lektoriranje. Posebej se zahvaljujem družini, ki mi je vedno stala ob strani in me spodbujala. Za moralno podporo tekom celega študija se zahvaljujem vsem študijskim kolegom in prijateljem.

Miloš Lukić, 2017

"If you can't measure it,
you can't improve it."

— Peter Drucker

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Pregled področja **5**

2.1 Obstoječi pristopi za evalvacijo poslovnih procesov 5

2.2 Evalvacija poslovnih procesov z uporabo dnevniških zapisov . 13

2.3 Orodja za nadzor različic 19

3 Opis evalvacijskega pristopa **23**

3.1 Načrtovanje 24

3.2 Evalvacija 27

3.3 Analiza dnevniških zapisov 34

3.4 Potrjevanje in poročanje rezultatov 49

4 Študija primera **55**

4.1 Opis podjetij 56

4.2 Rezultati evalvacij v podjetjih 64

4.3 Skupni rezultati 88

5 Sklepne ugotovitve **93**

Seznam uporabljenih kratic

kratica	angleško	slovensko
VCS	version control system	sistem za nadzor različic
ERP	enterprise resource planning	celovita programska rešitev
CMMI	capability maturity model integration	model za vrednotenje zrelosti organizacije
DOI	diffusion of innovation	difuzija inovacij
PCI	perceived characteristics of innovating	zaznane karakteristike inovacij
TRA	theory of reasoned action	teorija razumne akcije
TDD	test driven development	testno vodeni razvoj
SDM	software development methodology	metodologija razvoja programske opreme
TAM	technology acceptance model	model sprejetosti tehnologije
TPB	theory of planned behaviour	teorija načrtovanega vedenja
KDD	knowledge discovery in databases	odkrivanja znanj iz podatkovnih baz

Povzetek

Naslov: Evalvacija procesov razvoja programske opreme s pomočjo dnevniških zapisov razvojnega okolja

Obstoječi pristopi za ocenjevanje procesov razvoja programske opreme uporabljajo povratne informacije zaposlenih kot glavni vir podatkov. V tem magistrskem delu nas je zanimalo, kako bi vpeljava dnevniških zapisov kot dodatnega vira podatkov vplivala na rezultate evalvacije. Najprej smo pregledali obstoječe pristope, ki pri ocenjevanju upoštevajo sociološki, tehnični in ekonomski vidik poslovnih procesov. Na podlagi obstoječih pristopov smo razvili novega, v katerem poleg ocen, ki jih podajo zaposleni, upoštevamo tudi kvantitativne podatke iz dnevniških zapisov orodij, ki jih razvijalci uporabljajo pri opravljanju svojega dela. Zasnovani pristop smo preizkusili s pluralno študijo primera v treh podjetjih. Podobnost podjetij nam je omogočila preverjanje replikacije rezultatov. Ocene poslovnih procesov, ki smo jih pridobili z upoštevanjem obeh virov, smo posredovali vodstvom podjetij, ki so potrdila njihovo uporabnost. Skozi študijo primera smo ugotovili, da so za vodstvo podjetja ocene pridobljene na podlagi analize dnevniških zapisov uporabne, ker omogočajo zmanjšanje morebitnih odstopanj ocen razvijalcev od dejanskega stanja.

Ključne besede

evalvacija poslovnih procesov, dnevniški zapisi, razvojno okolje

Abstract

Title: Evaluation of software development processes using development environment logs

Existing approaches for evaluation of software development processes use employee feedback as a main source of data. We were interested in assessing the possible benefit of using event logs as an additional source of data in software development process evaluation. In this master's thesis, we reviewed existing approaches that take into account the sociological, technical and economic aspects of business processes. We have developed a new approach which in addition to employee feedback introduces the use of quantitative event log data generated by development environment tools to improve the accuracy of evaluation. We tested our approach in three companies using multiple case study method. Similarity of cases allowed us to verify the replication of the results. The management of all three companies confirmed that our approach provided useful information about the efficiency and acceptance of development processes. We confirmed that the use of event log data in evaluation is useful because it helps reduce possible deviation of developers' estimates from actual situation.

Keywords

business process evaluation, event logs, development environment

Poglavje 1

Uvod

Merjenje učinkovitosti podjetja je ključnega pomena za ugotavljanje njegovega napredka [11]. Vodstvo lahko s pomočjo meritev oceni trenutno stanje podjetja, ki služi kot podlaga za sprejemanje nadaljnjih odločitev. Izvajanje in interpretacija meritev predstavljata strošek za podjetje, zato je treba izbrati merila, ki prinašajo dodano vrednost pri odločanju. Izbira meril se razlikuje glede na vidik poslovanja, ki ga želimo oceniti. Ocenjujemo lahko učinkovitost zaposlenih, izdelka, poslovnih procesov ipd. V tem delu smo se osredotočili na vidik poslovnih procesov podjetja.

Poslovni procesi so množica logično povezanih postopkov in aktivnosti v podjetju, katerih izid je načrtovan izdelek ali storitev [53]. Delimo jih na primarne in podporne procese. Primarni poslovni procesi so tisti, katerih rezultat je izdelek ali storitev neposredno namenjena strankam, podporni procesi pa proizvajajo izdelke ali storitve, ki so strankam nevidne, a so ključne za uspešno poslovanje. Podrobneje smo se posvetili podjetjem, katerih primarni poslovni procesi so povezani z razvojem programske opreme (razvojni procesi).

Evalvacije poslovnih procesov so postale pogost predmet raziskav, saj organizaciji omogočajo vpogled v uporabnost in izkoristek njenih poslovnih procesov. Na nekaterih področjih, kot je denimo proizvodnja avtomobilov, se pri evalvaciji upoštevajo rezultati poslovnih procesov [21], na primer število

proizvedenih avtomobilov v časovnem intervalu. Procesi razvoja programske opreme so fleksibilni in niso jasno zastavljeni, kar onemogoča uporabo splošnih meril [21] za določanje rezultatov. Zato veliko število uveljavljenih pristopov [67, 68, 40, 27] za ocenjevanje poslovnih procesov upošteva povratne informacije zaposlenih, ne pa tudi meritev rezultatov poslovnih procesov. Kljub temu, da so se tovrstne raziskave v različnih študijah izkazale kot učinkovite [67, 68, 40], nas zanima, kako bi upoštevanje omenjenih meril vplivalo na rezultate. Subjektivne meritve, ki temeljijo na povratnih informacijah zaposlenih, so uporabne za ocenjevanje odnosa zaposlenih do procesov. Za ocenjevanje pogostosti izvajanja ali rezultatov določenega procesa je bolj primerno upoštevati objektivne meritve [61], pridobljene s pomočjo ustreznega orodja. Objektivna merila nimajo neposrednega vpliva človeške pristranskosti in se zato uporabljajo na vseh področjih, kjer so dostopna [21, 53, 9]. Pri razvoju programske opreme se je treba za definiranje meril poglobiti v tehnično plat razvoja ter razumeti procese in orodja, ki jih razvijalci uporabljajo. To lahko poveča čas in vire podjetja potrebne za izvedbo evalvacije. Izziv je torej ugotoviti, ali uvedba objektivnih meril poveča kakovost rezultatov evalvacije in ali je razlika dovolj velika, da odtehta povečano zahtevnost izvedbe.

Meritve izvajanja poslovnih procesov lahko dobimo iz dnevniških zapisov programske opreme, ki so se pojavili zaradi potrebe po lažji diagnostiki v primeru programskih napak. Da bi ugotovili, pri katerih dogodkih je prišlo do napake, so začeli beležiti tudi uporabniške aktivnosti. Z razvojem podatkovnega rudarjenja, ki omogoča analitiko velikega števila podatkov, so se dnevniški zapisi uveljavili kot dodaten vir informacij o poslovnih procesih. Danes skoraj vse programske rešitve, vključno z orodji za podporo pri razvoju programske opreme, shranjujejo zapise o uporabniški aktivnosti v dnevnike [63]. Za pridobitev informacij o razvojnih procesih smo se osredotočili na orodja, ki jih tipično uporabljajo razvijalci programske opreme. Pri pridobivanju in analizi dnevniških zapisov razvojnega okolja se pojavi veliko izzivov. Najbolj opazna sta standardizacija oblike [46, 71] in dostopnost

dnevniških zapisov [57]. Orodja za nadzor različic (angl. *Version Control System* - VCS), ki hranijo izvorno kodo, zgodovino hroščev izdelka, komunikacijske arhive in sezname zahtev, so danes postali samoumevni del vsakega večjega projekta. Nabor orodij, ki jih razvijalci uporabljajo, se lahko med podjetji oziroma tudi znotraj enega podjetja razlikuje. Upoštevanje in povezovanje informacij iz vseh omenjenih sistemov bi nam omogočilo celovitejšo sliko aktivnosti razvijalcev. Težava se pojavi, ker lahko razvijalci pri svojem delu uporabljajo širok nabor različnih orodij z različnimi oblikami dnevniških zapisov. To lahko privede do težav pri pomenskem povezovanju zapisov, saj se pogosto uporabljajo različne oznake za razvijalce ali aktivnosti. Oviro predstavlja tudi pomanjkanje dnevniških zapisov za določene aktivnosti. Ni namreč nujno, da orodje beleži vse aktivnosti uporabnika. V takih primerih lahko poskusimo pridobiti sledi izvajanja na podlagi drugih, povezanih aktivnosti. Tako dostopnost kot stopnja standardiziranosti dnevniških zapisov neposredno vplivata na čas potreben za izvedbo evalvacije.

Za merjenje izvajanja poslovnih procesov je treba preveriti ustreznost že obstoječih orodij za analizo dnevniških zapisov. V primeru, da izbrana merila niso podprta z obstoječimi orodji, je treba ustvariti novo orodje, ki jih bo podpiralo. Pri pridobivanju meritev si lahko pomagamo tudi z rudarjenjem procesov. To je tehnika pridobivanja informacij o poslovnih procesih iz dnevniških zapisov [65], ki se je izkazala kot uporabna v podobnih raziskavah [37, 15, 56].

Cilj tega magistrskega dela je razviti pristop za evalvacijo procesov razvoja programske opreme, ki upošteva tudi rezultate analize dnevniških zapisov. Za ugotavljanje uporabnosti analize dnevniških zapisov pri evalvaciji smo izbrali obstoječi pristop za evalvacijo poslovnih procesov [67, 68], ki se je v praksi izkazal kot uporaben, in ga razširili z analizo dnevniških zapisov. Opravili smo študijo primera [73] v treh podjetjih, ki se ukvarjajo z razvojem programske opreme, z namenom preverjanja uspešnosti pristopa. Razvojne procese so najprej ocenili zaposleni v podjetjih. Nato smo pridobili tudi ocene na podlagi meritev iz dnevniških zapisov. Ocene obeh virov smo upoštevali

pri končni oceni. Vpliv meritev na končno oceno in dodatni čas, potreben za pridobivanje teh meritev, smo upoštevali pri oceni uporabnosti analize dnevniških zapisov. Končne rezultate smo preverili z vodstvi podjetij, ki so potrdila njihovo uporabnost pri prepoznavanju učinkovitosti in sprejetosti razvojnih procesov. Glavni prispevki dela so zasnova pristopa za ocenjevanje razvojnih procesov z uporabo analize dnevniških zapisov, preizkus pristopa v praksi in potrditev uporabnosti pristopa z vodstvom podjetja.

V nadaljevanju je predstavljena struktura tega dela. V tem poglavju sta opisana motivacija in izzivi pri vključitvi dnevniških zapisov v evalvacijo poslovnih procesov. Definirani so tudi cilji in prispevki tega magistrskega dela. V drugem poglavju je opisan pregled literature treh ključnih področij. Prvo področje obsega pregled pristopov za vrednotenje poslovnih procesov v organizacijah. Drugo področje obsega pregled stanja na področju analize dnevniških zapisov. Tretje področje pa vsebuje pregled orodij za nadzor izvorne kode in njihove povezanosti s procesi razvoja programske opreme. V tretjem poglavju je na podlagi pregledane literature opisan izbran pristop za vrednotenje poslovnih procesov in uporabljen kot osnova za opredelitev našega razširjenega pristopa, ki upošteva tudi rezultate analize dnevniških zapisov. Razširjeni pristop je razčlenjen na štiri dele. V četrtem poglavju je predstavljena študija primera v okviru katere smo razviti pristop preizkusili v treh podjetjih. Na začetku poglavja so opisana vsa tri podjetja. Sledi pregled rezultatov, ki so razdeljeni na rezultate vrednotenja razvojnih procesov in rezultate vrednotenja koristnosti posameznih meritev, pridobljenih z analizo dnevniških zapisov. V zadnjem poglavju smo podali sklepne ugotovitve in ključne prispevke magistrskega dela. Povzeli smo tudi omejitve in smernice za nadaljnje delo.

Poglavje 2

Pregled področja

Poglavje je namenjeno pregledu literature s področij evalvacije poslovnih procesov, analize dnevniških zapisov in orodij za nadzor različic. Največji poudarek je na literaturi, ki se ukvarja z razvojem ogrodij za vrednotenje poslovnih procesov, kar je osrednja tema tega dela. Ker je cilj podkrepiti rezultate evalvacije s podatki, pridobljenimi z analizo dnevniških zapisov, smo pregledali tudi literaturo s področja pridobivanja znanj iz dnevniških zapisov. Pregledali smo tudi orodja za nadzor izvorne kode in njihove značilnosti, saj so omenjena orodja vir dnevniških zapisov, ki jih analiziramo.

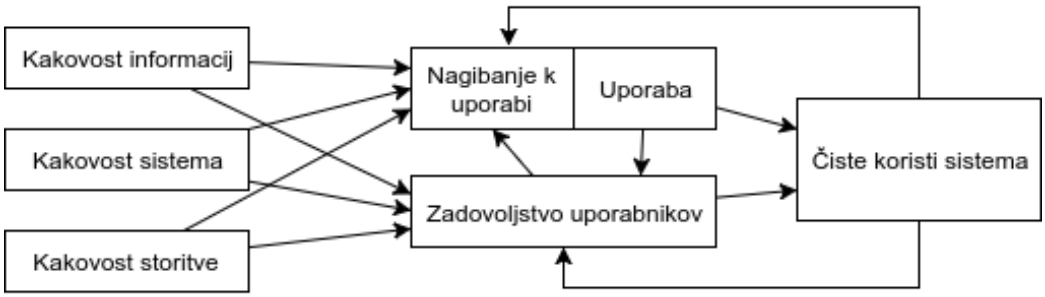
2.1 Obstoječi pristopi za evalvacijo poslovnih procesov

Procesi razvoja programske opreme se med različnimi organizacijami pogosto razlikujejo. Pomanjkanje konvencij, ki bi jih razvijalci morali upoštevati, je neredko razlog za nižjo produktivnost [12, 47]. Metodologija razvoja programske opreme je v delu [12] definirana kot množica konvencij, ki določajo različne vidike razvoja. Pogosto lahko tudi slabo zastavljene oziroma neustrezne metodologije vodijo k zmanjšani produktivnosti organizacije. Zaradi potrebe po definiranju in razumevanju razvojnih procesov znotraj organizacije je bilo razvitih več evalvacijskih modelov. V nadaljevanju je naštetih in

na kratko opisanih nekaj najbolj vplivnih in pomembnih modelov in teorij.

Capability Maturity Model Integration (CMMI) je eden izmed bolj prepoznavnih modelov, katerega namen je ocenjevanje zrelosti razvojnega procesa [11]. Model omogoča vrednotenje stopnje zrelosti poslovnih procesov v organizacijah. Predstavljenih je pet stopenj, višja kot je stopnja, bolj so razvojni procesi definirani, merljivi in deležni izboljšav. Prepoznavanje, definiranje in izboljševanje procesov lahko organizacijam pomaga doseči višje stopnje zrelosti. Doseganje višjih stopenj je odvisno od vzpostavitve meritev produktivnosti in kakovosti ter osredotočanja na izboljšanje poslovnih procesov. Izvedba evalvacije poslovnih procesov lahko pripomore k večji zrelosti podjetja. Evalvacija omogoča podjetju merjenje učinkovitosti in sprejetosti poslovnih procesov, kar služi kot podlaga za njihovo izboljšanje.

Model uspešnosti informacijskih sistemov [16], ki sta ga definirala Delone in McLean je ena izmed najbolj vplivnih teorij na področju raziskovanja informacijskih sistemov. Teorija je bila razvita leta 1992 in je od takrat deležna več posodobitev, njen namen pa je razložiti uspešnost informacijskega sistema s pomočjo definiranja šestih med seboj povezanih kritičnih dimenzij evalvacije. Model je prikazan na sliki 2.1.



Slika 2.1: Model Delone McLean za ocenjevanje uspešnosti

Povezane kritične dimenzije so kakovost informacij, sistema in storitve, nagibanje k uporabi in dejanska uporaba, zadovoljstvo uporabnikov ter čiste koristi sistema. Dimenzije ocenjevanja uspešnosti informacijskih sistemov se lahko delno prenesejo tudi na ocenjevanje poslovnih procesov, saj so infor-

macijski sistemi zgolj orodja za lažje izvajanje teh procesov.

Rogersova **teorija difuzije inovacij** [49] je ena izmed najbolj uveljavljenih teorij, ki se ukvarja s stopnjo širjenja inovacij (angl. *Diffusion of Innovations* - DOI). Teorija stremi k obrazložitvi kako in zakaj ter s kakšno hitrostjo se nove ideje in tehnologije (inovacije) širijo. Difuzijo opredeli kot proces, s katerim se inovacija čez čas razširi skozi socialni sistem. Glavni poudarek teorije je na sociološkem vidiku sprejetosti. Rogers definira štiri vplive na širitev ideje. Vplivi so inovacija sama, komunikacijski kanali, čas in socialni sistem. Teorija je splošno zastavljena in je bila kot taka uporabljena za podlago številnim raziskavam na različnih področjih [20]. Glavne karakteristike inovacije po teoriji DOI so naslednje:

- **Relativna prednost** meri, kakšno prednost zaznavajo njeni potencialni uporabniki v primerjavi z obstoječim stanjem.
- **Združljivost** meri stopnjo skladnosti med inovacijo in uporabniki na podlagi obstoječih vrednot in izkušenj.
- **Zahtevnost** oziroma preprostost za uporabo je stopnja, ki meri zaznano težavnost uporabe inovacije s strani uporabnikov.
- **Možnost preizkušanja** meri, kako preprosto lahko potencialni uporabniki inovacijo preizkusijo pred sprejetjem odločitve o njeni uporabi.
- **Možnost opazovanja** nam pove, kako je rezultat inovacije viden možnim uporabnikom.

Teorija DOI je bila uporabljena kot osnova za ogrodje **Zaznane karakteristike inovacij** (angl. *Perceived Characteristics of Innovating* - PCI) [2]. PCI razširi dejavnike inovacij, naštete pri DOI, tako, da relativni prednosti, združljivosti, zahtevnosti in možnosti preizkušanja doda še naslednje štiri karakteristike:

- **Predstavljivost rezultatov**, ki meri zmožnost zaznavanja rezultatov (nadomesti karakteristiko možnosti opazovanja, definirano v DOI).

- **Podoba** je stopnja, do katere uporaba inovacije izboljšuje posameznikovo podobo oziroma socialni status.

- **Vidljivost** je stopnja, do katere lahko drugi opazujejo uporabnike inovacije med dejansko uporabo.

- **Prostovoljnost** je stopnja, do katere uporabnik vidi inovacijo kot predpisano oziroma poljubno za uporabo.

Raziskovalci na področju inovacij poslovnih procesov in metodologij razvoja programske opreme (angl. *Software Development Methodology* - SDM) obravnavajo metodologijo oziroma njene dele kot inovacijo, razvijalce pa kot uporabnike inovacije [22].

Teorija razumne akcije (angl. *Theory of Reasoned Action* - TRA) je poleg Rogersove teorije ena izmed najbolj pomembnih za ugotavljanje sprejetosti tehnologij [19]. Teorijo sta razvila Martin Fishbein in Icek Ajzen leta 1967 kot nadgradnjo teorije informacijske integracije [5]. TRA se uporablja za napovedovanje obnašanja posameznikov na podlagi svojih obstoječih odnosov in vedenjskih namenov. Posameznikova odločitev za izvajanje določene akcije temelji na rezultatih oziroma posledicah, ki jih posameznik pričakuje po izvajanju.

Teorija razumne akcije je bila uporabljena kot osnova za **Model sprejetosti tehnologij** (angl. *Technology Acceptance Model* - TAM) [47] in njegovo nadgradnjo Model sprejetosti tehnologij 2 (angl. *Technology Acceptance Model 2* - TAM2) [70]. TAM je kot dva osrednja dejavnika za uporabnikovo pripravljenost uporabe nove tehnologije definiral zaznano uporabnost (angl. *percieved usefulness*) in zaznano enostavnost uporabe (angl. *perceived ease of use*). Zaznana uporabnost je dimenzija prepričanosti uporabnika, da bo določena aktivnost izboljšala njegovo produktivnost, zaznana enostavnost uporabe pa je prepričanje posameznika o enostavnosti izvedbe določene aktivnosti. Model TAM2 je ogrodju TAM dodal še dva dejavnika, ki sta namenjena zaznavanju prostovoljnosti posameznikovih aktivnosti. Subjektivna norma (angl. *subjective norm*) določa obseg posameznikovega dojemanja o

pričakovanjih okolja glede njegovih dejavnosti ter mero podrejanja omenjenim pričakovanjem.

Teorija načrtovanega vedenja (angl. *Theory of Planned Behaviour* - TPB) [3] je zasnovana na psihološkem vidiku sprejemanja tehnologij, poudarek pa je na vplivu prepričanja posameznika na njegovo obnašanje. Poleg že omenjene subjektivne norme, sta definirana še zaznana kontrola vedenja (angl. *perceived behavioural control*) - posameznikovo zavedanje svojih zmožnosti za določeno obnašanje in odnos do obnašanja (angl. *attitude towards behaviour*), ki predstavlja zaznano stopnjo oziroma oceno uspešnosti obnašanja s strani uporabnika.

Omenjene teorije in ogrodja se ukvarjajo z razlogi za sprejetosti oziroma nesprejetost tehnologij, ne upoštevajo pa tehnične dovršenosti in kakovosti inovacij. Kot osnova za nadgradnjo v tej magistrski nalogi je bil izbran pristop [67], ki zajema tako sociološki kot tudi tehnični vidik poslovnih procesov. Raziskovalci na področju sprejetosti SDM tipično opazujejo SDM kot celoto ali pa je predmet opazovanja le en proces. V pristopu [67] je SDM definiran kot skupek različnih aktivnosti, orodij in nalog, ki so uporabljene za določen del razvoja izdelka. Pristop za evalvacijo SDM [67] je bil preizkušen v več podjetjih. Metodologija je bila razdeljena na posamezne enote. Vsaka enota je bila nato ocenjena iz sociološkega vidika in na novo vpeljanega **tehničnega vidika**, ki upošteva ustreznost aktivnosti za tehnološke potrebe projekta. Pristop [67] definira štiri skupine karakteristik, ki merijo razloge za trenutno stanje tehnične sprejetosti in karakteristike, ki merijo trenutno stanje. Spodaj so naštete vse štiri skupine.

Karakteristike, ki merijo tehnično ustreznost:

- Ustreznost za projekt in sistem predstavlja stopnjo, do katere element ustreza različnim projektom in sistemskim parametrom, kot so velikost, zahtevnost, prioriteta, tip, itd.

- Ustreznost za razvojno ekipo določa, do katere mere je določeni element ustrezen izkušnjam in znanju razvojne ekipe - iz vidika tehničnega vodstva.

- Ustreznost za stranko - stopnja, do katere določen element ustreza zahtevam in potrebam strank.
- Skladnost z modernimi razvojnimi pristopi - določa stopnjo, do katere je ustrezen element v koraku s časom z moderno tehnologijo, tehnikami in priporočili na svojem področju.

Karakteristike, ki merijo tehnično združljivost:

- Združljivost z informacijskimi tehnologijami in programskim okoljem meri, do katere stopnje je določeni element skladen s tehnologijami in programskimi okolji, uporabljenimi za razvoj na določenem projektu.
- Združljivost z internimi standardi SDM - stopnja, do katere se določen element sklada z internimi standardi podjetja.
- Združljivost s splošnimi standardi meri, do katere stopnje določeni element SDM ustreza splošno definiranim standardom, kot so standardi kodiranja, arhitekture, vzorcev, modeliranja itd.

Karakteristike, ki merijo lastnosti prilagodljivosti:

- Prilagodljivost tehničnim potrebam projekta je stopnja, do katere se lahko posamezni element prilagodi tehničnim potrebam projekta.
- Prilagodljivost potrebam uporabnikov je stopnja, do katere se lahko element prilagodi ustrezni stopnji znanja in izkušenj posameznih uporabnikov SDM.

Glavna karakteristika tehničnega vidika, ki meri trenutno stanje, je frekvenca priložnosti za uporabo elementa. Omenjena karakteristika meri, kako pogosto se pojavi priložnost za uporabo posameznega elementa, ne glede na to, ali je dejansko uporabljen.

Sledijo karakteristike, ki se ukvarjajo s posledicami uporabe posameznega elementa na različne vidike razvoja. Omenjeni vidiki razvoja so našteti v nadaljevanju.

- Sistem - vpliv na vzdrževanje, zahtevnost, uporabnost, zanesljivost.
- Projekt - poraba časa, virov organizacije, kakovost planiranja, sledenje itd.
- Uporabniki - razumevanje dolžnosti in odgovornosti, komunikacija, sodelovanje in podpora za izobraževanje.
- Organizacija - standardizacija, doseganje ciljev organizacije in izboljšanje ugleda organizacije.
- Stranke - izboljšanje zaupanja strank, zadovoljstvo strank z napredkom in splošnim vtisom organizacije.

Podobno kot tehnični vidik, ogrodje razdeli tudi sociološkega, in sicer na trenutno raven sociološke sprejetosti in na razloge zanjo.

Karakteristike trenutne ravni sociološke sprejetosti:

- Frekvenca uporabe v primeru ponujene priložnosti meri, kako pogosto uporabniki uporabijo element v primeru, da se pojavi priložnost za njegovo uporabo znotraj projekta.
- Konsistenca uporabe meri, v kakšni meri se razvijalci držijo navodil in pravil določenega elementa.

Karakteristike, ki merijo razloge za trenutno stanje sociološke sprejetosti so razdeljeni v štiri skupine, prvo sestavljajo zaznani atributi SDM:

- Relativna prednost elementa, karakteristika izvira iz DOI. V različnih delih je označena kot ena najbolj pomembnih karakteristik, ki pozitivno vplivajo na sprejetost SDM.
- Socialna združljivost je stopnja, do katere uporabniki zaznajo skladnost med elementom in njihovim znanjem, izkušnjami in potrebami.
- Zahtevnost je stopnja, ki določa, kako zahteven je element za uporabo iz vidika uporabnika.

Druga skupina so karakteristike atributov uporabnikov SDM:

- Izkušnje in znanje na področju programiranja, orodij, platform, tehnologij itd.
- Izkušnje in znanje na področju metodologij razvojnih procesov v kontekstu obravnavanega elementa.

Tretja skupina so karakteristike zaznanih atributov organizacije, mednje sodijo prostovoljnost (izpeljana iz PCI), subjektivna norma (enako kot pri TRA) in podpora vodstva, ki meri, koliko je vodstvo aktivno pri vpeljevanju in uporabi določenega elementa v projektih. Četrta skupina so zaznani atributi predstavitve SDM, vključujejo prestavljivost rezultatov (enako kot pri PCI), vidljivost (PCI) in dostopnost znanja o uporabi SDM elementa.

Tehnični vidik z ocenjevanjem tehnične ustreznosti elementov ustvari celovitejšo sliko ustreznosti procesov, vendar tudi v tem primeru manjka ekonomski pogled vodstva organizacije na procese. Kljub slabi oceni s tehničnega in sociološkega vidika je lahko izboljšava določenega elementa ekonomsko neustrezna glede na koristi, ki jih prinaša. Zato se je pojavila potreba po uvedbi **ekonomskega vidika** [68], ki je ključnega pomena za vodstvene in ekonomsko naravnane odločitve.

Za ocenjevanje vodstvenega oziroma ekonomskega vidika poslovnih procesov so raziskovalci tipično upoštevali čas, stroške in kakovost izvedbe [60]. Omenjena merila sestavljajo železni trikotnik, ki naj bi bil temelj ocenjevanja uspešnosti vodenja projektov. Izkazalo se je, da ima teorija železnega trikotnika več omejitev. V delu [6] sta čas in stroški predstavljena zgolj kot oceni v trenutku, ko se o projektu ve najmanj. Kakovost je predstavljena kot pojav, odvisen od zaznavanja ljudi, ta se pa lahko v času izvedbe projekta spremeni. Zato so v sodobnejših delih [68] razširili kriterij časa, ki po novem vključuje strateške cilje organizacije, kriteriju kakovosti pa je bila dodana meritev vrednosti izdelka za stranke in poslovne partnerje. Namesto stroškov se priporoča upoštevanje kriterija dodane vrednosti za lastnike in organizacijo [6]. Dodana vrednost za organizacijo je predstavljena kot naj-

boljši kriterij za merjenje učinkovitosti, saj upošteva celotno na novo ustvarjeno vrednost, ki jih povzroča določen element. V nekaterih primerih [68] se je izkazalo, da vodstvo pri določenih elementih ocenjevanja ne more podati ustrezne ocene dodane vrednosti. Zato so tudi v omenjenem pristopu uporabili stroške kot glavno merilo učinkovitosti. Eden izmed razlogov za to je, da so vodstva podjetij v preteklosti za ocenjevanje učinkovitosti poslovnih procesov vedno uporabljala kriterij stroškov.

2.2 Evalvacija poslovnih procesov z uporabo dnevniških zapisov

Dnevniški zapisi so sistemsko ustvarjeni zgodovinski podatki, ki odražajo aktivnosti sistema. Sistemi pogosto ustvarjajo veliko število dnevniških zapisov. Za pridobivanje koristnih informacij iz dnevniških zapisov so potrebni pristopi, ki omogočajo obdelavo in prepoznavanje vzorcev v veliki količini podatkov. Eden izmed omenjenih pristopov je podatkovno rudarjenje, katerega podmnožica je rudarjenje dnevniških zapisov.

2.2.1 Podatkovno rudarjenje

Podatkovno rudarjenje je sistematično iskanje informacij v veliki količini podatkov. Vključuje metode s področij strojnega učenja, statistike, prepoznavanja vzorcev, podatkovnih baz in skladišč, vizualizacij, algoritmov ter pridobivanja informacij [26]. Cilj podatkovnega rudarjenja je pridobiti informacije iz nabora podatkov in ga spremeniti v razumljivo strukturo za nadaljnjo uporabo. Podatkovno rudarjenje je definirano kot ena od petih stopenj v procesu odkrivanja znanj iz podatkovnih baz (angl. *Knowledge Discovery in Databases* - KDD) [26]. V omenjenem delu je proces KDD razdeljen na naslednje stopnje:

- Izbira
- Predprocesiranje

- Transformacija
- Podatkovno rudarjenje
- Interpretacija/evalvacija

Podatkovno rudarjenje je že uveljavljen proces na različnih področjih, kot so medicina, znanost, finance ipd. [17, 41, 51]. Kot poskus poenotenja definicije rudarjenja podatkov je nastal od dejavnosti neodvisen standardiziran proces podatkovnega rudarjenja (angl. *Cross Industry Standard Process for Data Mining* - CRISP-DM) [38], ki definira podatkovno rudarjenje kot postopek šestih stopenj:

1. Razumevanje poslovanja - v prvi fazi je poudarek na razumevanju ciljev projekta in zahtev iz poslovne perspektive, ter pretvorba poslovnih zahtev v definicijo problema iz perspektive podatkovnega rudarjenja.

2. Razumevanje podatkov - stopnja se začne z zbiranjem osnovnih podatkov in definira procese, usmerjene v razumevanje podatkov, njihove kakovosti, prepoznavanje zanimivih množic oziroma podmnožic podatkov, s katerimi lahko tvorimo hipotezo.

3. Priprava podatkov - ta stopnja definira vse procese sestavljanja končne množice podatkov iz osnovnega nabora, ki bo uporabljena kot vhod orodjem za modeliranje.

4. Modeliranje - v tej fazi so na končni množici podatkov uporabljene različne tehnike modeliranja. Vključuje tudi iskanje parametrov modela, za pridobitev optimalnih izhodnih vrednosti.

5. Evalvacija - v tej fazi je model že definiran, treba ga je še ovrednotiti. Cilj te stopnje je ugotoviti, kateri vidiki oziroma problemi niso uspešno pokriti z uporabo izbranega modela.

6. Postavitev modela - Po izbiri oziroma definiranju modela in pridobljenem znanju iz podatkov je treba to znanje organizirati in predstaviti na

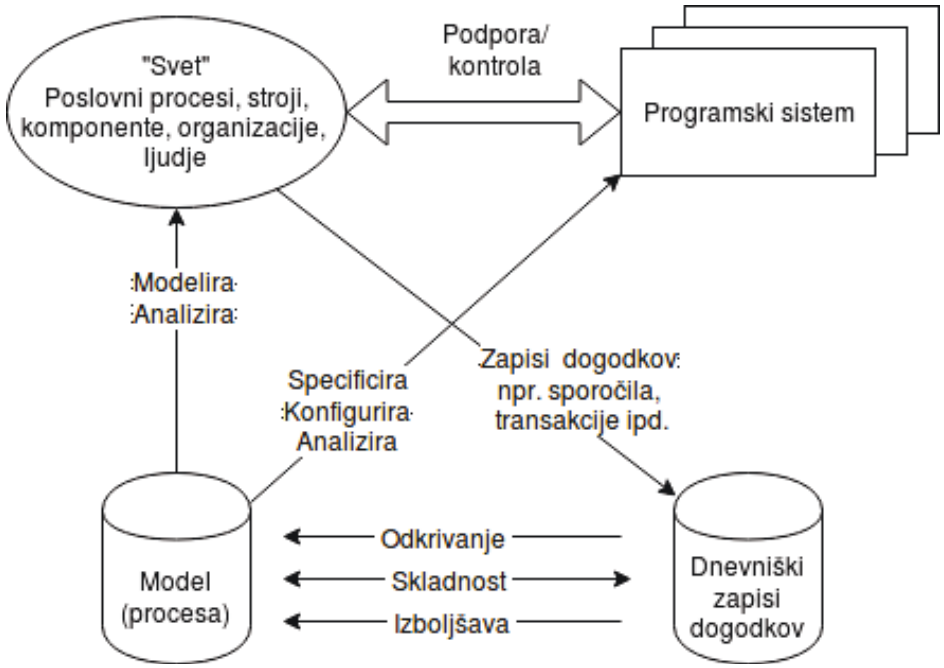
način, ki bo uporaben končni stranki. Rezultat tega procesa se lahko razlikuje po zahtevah - lahko je preprosto poročilo ali kompleksen periodičen proces pridobivanja rezultatov iz podatkov. Cilj je končni stranki omogočiti razumevanje uporabe modela.

Posebna vrsta podatkovnega rudarjenja je rudarjenje procesov, katerega cilj je pridobiti informacije o uporabniških aktivnostih iz dnevniških zapisov orodij.

2.2.2 Rudarjenje procesov

Rudarjenje procesov se uporablja za izboljšanje razumevanja in učinkovitosti poslovnih procesov. Pogoj za uspešno izvedbo rudarjenja procesov je prisotnost dnevniških zapisov [65], ki vsebujejo podatke o zgodovini izvajanj poslovnih procesov. V procese razvoja programske opreme so najbolj vpeta orodja za nadzor izvorne kode, zato je rudarjenje dnevniških zapisov omenjenih orodij pogost predmet raziskav [56, 15, 57]. Rudarjenje procesov se uporablja tudi na drugih področjih, kot so na primer zdravstvo [50], izobraževanje [51], poslovanje [42], energetika [41], ipd. Vsem raziskavam je skupno to, da uporabljajo podatke o zgodovini izvajanja poslovnih procesov za analizo in izboljšanje prihodnjih izvajanj. Povezanost rudarjenja procesov z ostalimi dejavniki je prikazana na sliki 2.2.

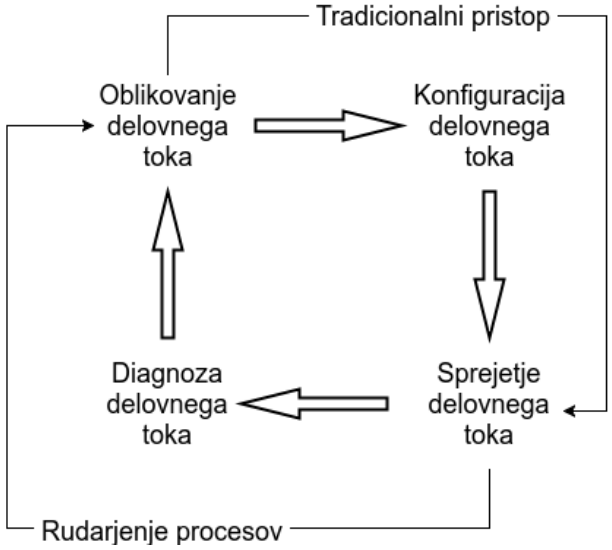
Zaradi dostopnosti pridobivanja uporabniških mnenj so pogoste raziskave, v katerih se uporabljajo tako uporabniška mnenja kot sistemski podatki uporabe za pridobivanje informacij o sprejetosti procesov [54]. Dnevniški zapisi beležijo sistemske dogodke, katerih vzrok je lahko sam sistem ali uporabnik. Iz tega sledi, da lahko iz dnevniških zapisov razberemo zaporedja aktivnosti razvijalca. Pri ponavljajočih se poslovnih procesih so pogosto definirana zaporedja aktivnosti, ki veljajo za najbolj optimalno pot do zastavljenega cilja. Izvajalci poslovnih procesov se morajo držati definiranih zaporedij, ki jih imenujemo delovni tok. Delovni tok je definiran kot natančno opredeljeno in informatizirano zaporedje opravil v aktivnosti, poslovnemu procesu ali zgolj



Slika 2.2: Shema povezanosti elementov pri rudarjenju procesov.

njegovem delu [64]. Definiran je tudi življenjski cikel delovnega toka [64], ki vsebuje štiri stopnje (slika 2.3).

Prva stopnja je oblikovanje delovnega toka, sledijo ji konfiguracija, sprejetje in diagnoza delovnega toka. Pri tradicionalnem pristopu se v stopnji oblikovanja zgradi model delovnega toka. Oblikovanje delovnega toka po navadi opravi vodstvo, svetovalci ali izkušeno osebje. Sledi stopnja konfiguracije, kjer je celoten delovni tok definiran in pripravljen za uporabo. V fazi sprejetja se primeri oziroma instance delovnega toka izvajajo v sistemu, ki je bil definiran v prejšnji stopnji. Uporaba delovnega toka omogoči zbiranje diagnostičnih informacij, ki so analizirane v stopnji diagnoze delovnega toka. Diagnostične informacije pomagajo pri izboljšanju oblikovanja delovnega toka, s čimer je življenjski cikel zaključen. Pri tradicionalnem pristopu je poudarek na prvih dveh stopnjah, manj pozornosti pa je na fazi diagnoze [64], pri kateri je treba sistematično zbirati podatke o uporabi in jih analizirati. Cilj rudarjenja procesov je obrniti postopek in na podlagi znanj pridobljenih iz zbranih podatkov analizirati in oblikovati poslovne procese.



Slika 2.3: Življenjski cikel delovnega toka.

Rudarjenje procesov temelji na predpostavki, da vsak dnevniški zapis vsebuje štiri ključne komponente [64]:

- **Aktivnost** - natančno definiran korak v procesu.
- **Primer** - ena izvedba procesa.
- **Izvajalec** - oseba, ki je začetnik ali izvajalec aktivnosti.
- **Časovni žig** - dogodki morajo biti časovno urejeni, zaradi potrebe po spremljanju zaporedja aktivnosti.

Rezultat rudarjenja je odvisen od opazovanega vidika. Lahko spremljamo vidik procesa, organizacije ali primera [65].

- **Vidik procesa** se osredotoča na urejanje zaporedja izvrševanja aktivnosti ter iskanja možnih poti med njimi, cilj rudarjenja tega vidika je iskanje vseh možnih poti, po navadi pa je izražen s Petrijevo mrežo [25] ali dogodkovno usmerjeno verigo procesov (angl. *Event-driven Process Chain* - EPC).

- **Vidik organizacije** je osredotočen na izvajalce aktivnosti, njihovo vpetost v ostale aktivnosti ter medsebojno povezanost. Cilj je klasificirati zaposlene skozi vloge in organizacijske enote ali pa poiskati relacije med posameznimi izvajalci, rezultat tega pa po navadi prikažemo s socialno mrežo.

- **Vidik primera** se osredotoča na lastnosti ene konkretne izvedbe delovnega toka, ki je že končana. Pri tem vidiku lahko spremljamo zaporedja procesov, njihove izvajalce in vire, ki so bili porabljeni ali proizvedeni med izvajanjem. Primer ene izvedbe je analiza odpravljanja določenega hrošča v izvorni kodi, kjer nas zanima kdo je zahteval popravilo, koliko razvijalcev je bilo potrebno za izvedbo in kako je potekala pot aktivnosti.

2.2.3 Obstoječa orodja za rudarjenje procesov

Za rudarjenje procesov obstaja veliko orodij, med katerimi jih večina obravnava zgolj vidik procesa [62, 72] (vidiki so opisani v poglavju 2.2.2). Izkazalo se je, da so omenjena orodja precej stara, neposodobljena in ne omogočajo obravnave ostalih dveh vidikov. Orodje Prom [65] je bilo razvito s podporo za obravnavo vseh treh vidikov. Izbrali smo ga za uporabo v študiji primera, saj je deležno številnih posodobitev in razširitev. Poleg omenjenih dejavnikov smo ugotovili, da ima to orodje boljšo uporabniško podporo in več navodil za uporabo kot podobna orodja. Pomembni lastnosti sta tudi odprtokodnost in možnost brezplačne uporabe.

Vhodni podatki za orodje Prom morajo biti podani v standardni obliki XES (Extensible event stream). XES je standard, ki definira pravila pri oblikovanju dnevniških zapisov. Cilj vpeljave standarda je določiti splošna pravila za oblikovanje dnevniških zapisov, ki bi se jih držali vsi proizvajalci programske opreme [71]. XES je nastal kot zamenjava za starejši format MXML [66]. Standard MXML je imel slabo podporo za razširitev dnevniških zapisov z dodatnimi atributi [71]. Posledično so se pri vpeljavi standarda

MXML pojavili problemi, ki so privedli do nastanka XES standarda. Enotna pravila pri oblikovanju dnevniških zapisov olajšajo izvajanje rudarjenja procesov, saj dnevniških zapisov ni treba preoblikovati.

2.3 Orodja za nadzor različic

Sistemi oziroma orodja za nadzor različic (angl. *Version Control System* - VCS) so namenjena upravljanju in nadziranju sprememb v zbirkah podatkov, kot so programi, dokumenti, spletne strani itd. V kategorijo orodij za nadzor različic po navadi štejemo specializirane aplikacije, katerih namen je spremljanje sprememb v zbirkah podatkov, kljub temu, da je nadzor različic pogosto vključeno kot funkcionalnost v raznovrstna orodja. Najbolj znani VCS so orodja za nadzor izvorne kode.

2.3.1 Pregled orodij za nadzor izvorne kode

Z naraščajočo kompleksnostjo izvorne kode programske opreme se poveča tudi potreba po njeni organiziranosti [58]. Danes imajo razvijalci na voljo veliko število različnih orodij za nadzor izvorne kode, kot so na primer Subversion, Git, Mercurial in podobni. Omenjena orodja spremljajo tekstovne datoteke, ki vsebujejo izvorno kodo, dokumentacijo in testne podatke. Naloge tovrstnih sistemov so avtomatizacija shranjevanja, pridobivanja, ustvarjanja in prepoznavanja različic. Večina orodij za nadzor različic zaradi optimizacije hitrosti delovanja in porabe prostora v določeni različici shranjuje le spremembe datoteke v primerjavi s prejšnjo različico [58]. Eno izmed najbolj popularnih orodij za nadzor izvorne kode je Git. Razvil ga je Linus Torvalds kot podporno orodje pri razvoju Linux jedra. Je eden izmed prvih porazdeljenih sistemov za nadzor izvorne kode [1], ki so nastali kot odgovor na uveljavljena centralizirana orodja, katerih glavni predstavnik je Subversion (SVN) [10].

Orodja za nadzor različic beležijo zgodovino različnih procesov, najbolj pogosti primeri so zgodovina izvorne kode, zgodovina hroščev programa, ko-

munikacijski arhivi in zgodovina seznamov zahtev in njihovih stanj. Hranjenje tovrstnih podatkov organizacijam omogoča pridobivanje informacij iz virov, ki neposredno in objektivno odražajo delo posameznikov, skupin ali celotnega podjetja. Poleg shranjevanja zgodovine različic orodja pogosto beležijo tudi dnevniške zapise, ki odražajo uporabnikove aktivnosti. Posledica dostopnosti omenjenih zapisov je možnost uporabe podatkovnega rudarjenja na orodjih VCS [48, 75]. Raziskave na področju podatkovnega rudarjenja orodij VCS so pogosto opravljene z namenom olajšanja dela razvijalcem s pomočjo različnih tehnik avtomatiziranega iskanja morebitnih napak ali povzročiteljev napak [75]. Raziskav, ki se ukvarjajo z rudarjenjem VCS za potrebe spremljanja procesa razvoja programske opreme, ni veliko. Vzroki za tako stanje so kompleksnost, dostopnost in nestandardnost dnevniških zapisov sistemov za nadzor izvorne kode [52] in so podrobneje opisani v poglavju 3.3.

2.3.2 Povezanost razvojnih procesov z orodji za nadzor izvorne kode

Razvijalci s pomočjo orodij za nadzor izvorne kode izvajajo različne aktivnosti, katerih sledi se beležijo v dnevniške zapise. Z analizo teh zapisov lahko pridobimo meritve pogostosti izvajanja aktivnosti, njihovo zaporedje, trend, izvajalce ipd. Rezultati meritev odražajo dejansko izvajanje procesov razvoja programske opreme. Različne metodologije razvoja lahko različno vplivajo na rezultate meritev. V nadaljevanju so predstavljeni primeri, kako strategija podjetja vpliva na razvojne procese in katere so najbolj tipične aktivnosti razvojnih procesov, ki jih lahko izvajamo z orodji za nadzor izvorne kode.

Večina mladih podjetij, katerih izdelek še ni popolnoma definiran, na začetku teži k hitremu prototipiranju in iskanju funkcionalnosti izdelka [44], ki bi ustrezale ciljni skupini. Ko je izdelek že dobro definiran, pa je na vrsti zagotavljanje stabilnosti in kakovost izdelka ter njegova nadaljnja rast [44]. Z novimi strateškimi cilji organizacije se spremenijo tudi poslovni procesi, ki jih razvijalci izvajajo v podjetju. Na začetku je poudarek na dokazovanju različnih konceptov, hitrem razvoju in iskanju novih pristopov, medtem

ko so kakovost kode, stabilnost izdelka, testiranje in dokumentacija zapostavljeni [43]. Bolj kot je izdelek uveljavljen, bolj zrelo postaja podjetje, hkrati pa postajajo tudi procesi vedno bolj formalizirani. Prehod je pogosto postopen [44], kar pomeni, da nekateri razvijalci počasneje sprejmejo nove razvojne procese kot drugi, kljub veliki verjetnosti, da je upoštevanje teh procesov koristno tako za razvijalce kot za izdelek [34, 32]. Počasnejši prehod lahko poveča stroške dodatnega dela [35], ki nastanejo zaradi izbire preprostih pristopov namesto kakovostnejših, ker so slednji lahko bolj časovno zahtevni. Primer je zmanjševanje obsega testiranja z namenom čim hitrejše izdelave programske opreme. Kratkoročno lahko to pomaga podjetju razviti izdelek hitreje od konkurence in pridobiti večje število strank [60]. Dolgoročno pa manj kakovosten izdelek zahteva večje stroške vzdrževanja.

Orodja za nadzor izvorne kode so bila zasnovana tako, da razvijalcem omogočajo urejeno in strukturirano shranjevanje zgodovine sprememb [36, 13]. Razvijalcem ponujajo možnost vzporednega dela na različnih vejah kode, avtomatizirano združevanje različnih vej, označevanje različic z metapodatki in še veliko drugih funkcionalnosti. Poleg urejenosti zgodovine izvorne kode in njenih različic se na orodja VCS navezuje še več drugih razvojnih procesov [39] kot kot npr. testiranje, pregledi programske kode (angl. *peer review*) [48, 74], dokumentiranje ipd. Avtomatizirano testiranje pred shranjevanjem kode se lahko odraža v manjšem številu shranjenih različic, od katerih vsaka zajema smiseln nabor delujočih sprememb [4], strokovne recenzije se odražajo pri zahtevah za združevanje kode, ki jih zadolženi razvijalci sprejemajo ali zavrnejo. Eden izmed izzivov v tem delu je tudi prepoznavanje sledi izvajanja različnih aktivnosti s pomočjo dnevniških zapisov.

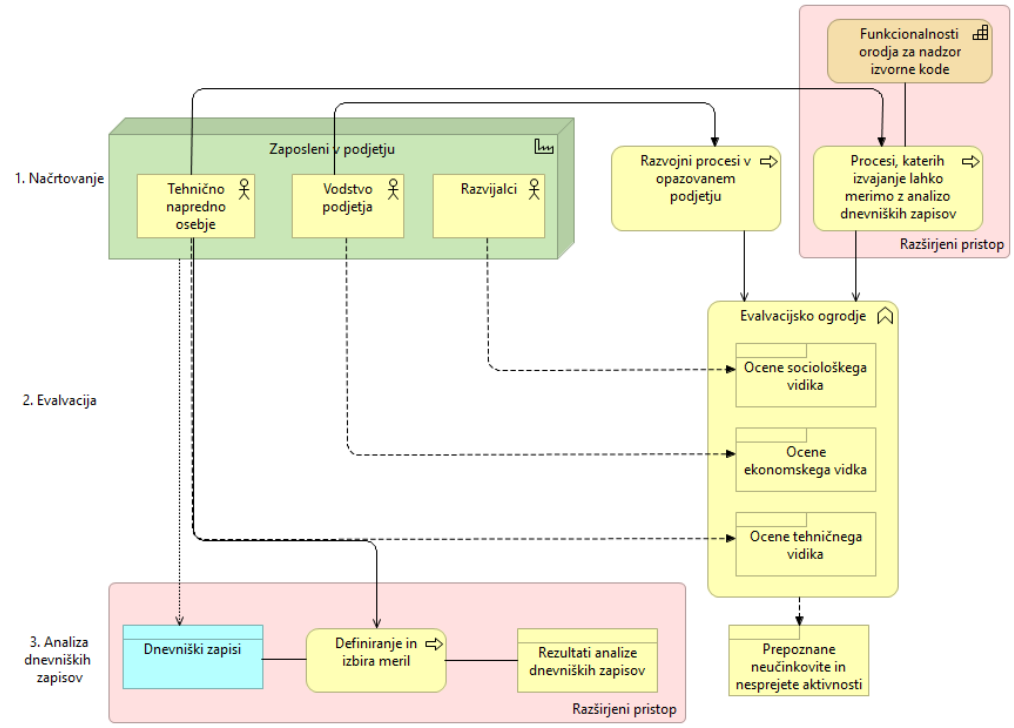
Poglavje 3

Opis evalvacijskega pristopa

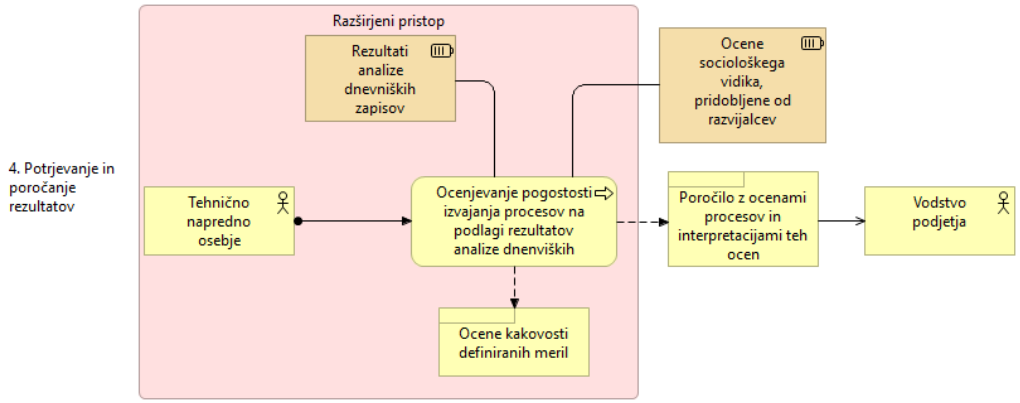
V tem poglavju je predstavljen pristop za evalvacijo razvojnih procesov s pomočjo analize dnevniških zapisov. Pristop temelji na področjih evalvacije poslovnih procesov, analize dnevniških zapisov in orodij za nadzor različic. Vsa tri področja so predstavljena v poglavju 2. Na osnovi pregledane literature smo za osnovo izbrali že obstoječa pristopa [67, 68], ki sta bila uspešno testirana v okviru študij primerov. Pristopa smo razširili z vključitvijo analize dnevniških zapisov v evalvacijo. Za potrebe te magistrske naloge smo se osredotočili na vrednotenje razvojnih procesov. Pristop je razdeljen na naslednje štiri dele:

- Načrtovanje
- Evalvacija
- Analiza dnevniških zapisov
- Potrjevanje in poročanje rezultatov

Diagram poteka pristopa je prikazan na slikah 3.1 in 3.2. Oba diagrama sta vizualizirana z jezikom ArchiMate [23], ki se uporablja za modeliranje poslovnih komponent. Z rdečo barvo so označeni viri in koraki, ki smo jih na novo vključili v pristop.



Slika 3.1: Diagram poteka prvih treh delov pristopa.



Slika 3.2: Diagram poteka četrtega dela pristopa.

3.1 Načrtovanje

V tem delu pristopa pridobimo informacije o podjetju in poslovnih procesih, ki se v njem izvajajo. Načrtovanje izhaja iz pristopa, ki sta ga razvila

Vavpotič in Bajec [67]. Omenjeni pristop smo razširili tako, da vključuje tudi raziskovanje primernosti uporabe dnevniških zapisov pri evalvaciji. Na podlagi ugotovitev se odločimo, ali je smiselno uporabiti analizo dnevniških zapisov v evalvaciji.

Načrtovanje smo razdelili na naslednje tri korake:

1. Ugotavljanje obsega in smiselnosti izvedbe evalvacije
2. Prepoznavanje vlog zaposlenih in tehnično naprednega osebja
3. Izbira osnovnih elementov za ocenjevanje in meril za analizo dnevniških zapisov

V prvem koraku je treba prepoznati osnovne lastnosti podjetja in poslovnih procesov, ki se v njem odvijajo. Za ugotavljanje obsega evalvacije potrebujemo informacije, kot so velikost razvojne ekipe, narava in obseg projektov ter stopnja formaliziranosti razvojnih procesov. Osnovne informacije o podjetju in poslovnih procesih pridobimo od vodstva podjetja. Za določanje smiselnosti izvedbe evalvacije je treba ugotoviti motiviranost in pripravljenost zaposlenih za sodelovanje. Zanima nas tudi, katera orodja se v podjetju uporabljajo in kako so povezana z razvojnimi procesi.

V drugem koraku planiranja moramo zaposlenim, ki sodelujejo v evalvaciji, določiti vloge. Vsaka vloga ima določeno vedenje in odgovornosti znotraj podjetja [31]. Določeno vlogo ima lahko eden ali več zaposlenih. Tipične vloge v ekipah, ki se ukvarjajo z razvojem programske opreme, so programer, sistemski administrator, oblikovalec in tester. Na podlagi prepoznanih vlog lahko določimo, katere aktivnosti razvojnih procesov bo posameznik ocenjeval. Poleg vlog je v tem koraku treba prepoznati tudi tehnično napredno osebje. V to skupino spadajo zaposleni, ki imajo poglobljeno znanje o razvojnih procesih in so seznanjeni s sodobnimi trendi na področju teh procesov. Poleg poznavanja procesov morajo dobro poznati tudi orodja, ki se uporabljajo pri izvajanju procesov. Zaželeno je, da razumejo strukturo in vsebino dnevniških zapisov, ki jih uporabljena orodja beležijo. V primeru,

da niso seznanjeni s strukturo in vsebino dnevniških zapisov, jim lahko pri tem pomagajo razvijalci, ki to znanje posedujejo. Tehnično napredno osebje bo zadolženo za ocenjevanje učinkovitosti posameznih aktivnosti v razvojnih procesih. Za potrebe našega pristopa smo zadolžitve tehnično naprednega osebja razširili tudi na ocenjevanje primernosti dnevniških zapisov.

Zadnji korak načrtovanja je izbira elementov za ocenjevanje. V prvem koraku načrtovanja smo prepoznali razvojne procese v podjetju. Vsak razvojni proces je sestavljen iz določenega nabora aktivnosti. Pri izvajanju posamezne aktivnosti si lahko zaposleni pomagajo z različnimi orodji. Prepoznane razvojne procese je treba razčleniti na posamezne aktivnosti. V tem delu smo se omejili izključno na ocenjevanje razvojnih procesov, kar nam pri enaki časovni zahtevnosti omogoča bolj podrobno razumevanje procesov, kot če bi želeli ocenjevati vse poslovne procese v podjetju. Analizo razvojnih procesov [67] želimo poglobiti z upoštevanjem analize dnevniških zapisov. Posledično moramo pri izbiri osnovnih elementov za ocenjevanje upoštevati tudi dostopnost in podrobnost dnevniških zapisov. Za osnovne evalvacijske enote moramo izbrati aktivnosti, katerih sledi izvajanja so vsaj delno prisotne v dnevniških zapisih. Dopuščamo tudi izbiro takih evalvacijskih enot, katerih sledi izvajanja lahko pridobimo posredno. Tehnično napredno osebje lahko v tem primeru prepozna različna odstopanja, kot so na primer pomanjkanje pogojev za izvedbo ključne aktivnosti, trend zmanjševanja izvajanja določene aktivnosti, kljub vse večjemu številu priložnosti za izvedbo, neustrezna zaporedja izvajanj ali napake pri izvajanju ipd.

V zadnjem koraku želimo pridobiti tudi informacije o orodjih in njihovih dnevniških zapisih. Treba je ugotoviti primernost uporabe dnevniških zapisov v evalvaciji, pri čemer nam lahko pomaga tehnično napredno osebje. Za ugotavljanje primernosti je treba preveriti dostopnost, kompleksnost in kakovost dnevniških zapisov. Dostopnost dnevniških zapisov je lahko odvisna od orodja ali privolitve organizacije. Preveriti moramo, ali orodja beležijo dnevniške zapise in ali nam organizacija omogoča vpogled v te zapise. Kompleksnost dnevniških zapisov je odvisna od njihove oblike in raznovrstnosti.

Oblika dnevniških zapisov se lahko med različnimi orodji razlikuje. Več kot imamo različnih oblik dnevniških zapisov, bolj časovno zahtevna postane njihova analiza. Časovna zahtevnost se poveča zaradi potrebe po preoblikovanju dnevniških zapisov v standardno obliko, ki jo podpirajo orodja za analizo dnevniških zapisov (kot je opisano v poglavju 2.2.3). Ugotavljanje kakovosti, dostopnosti in kompleksnosti dnevniških zapisov je bolj podrobno opisano v poglavju 3.3. Ko so našteti dejavniki raziskani, se je treba s tehnično naprednim osebjem uskladiti glede izbire meril za analizo dnevniških zapisov. Merila so predstavljena v poglavju 3.3.3.

Želimo izbrati orodja, katerih dnevniški zapisi smiselno odražajo določen sklop razvojnih procesov. Vsako orodje se uporablja z namenom olajšanja izvedbe enega ali več razvojnih procesov. Za boljše razumevanje konteksta izvajanja razvojnih procesov je treba imeti dostop tudi do dnevniških zapisov izvajanja podpornih procesov. Za analizo dnevniških zapisov celotnega nabora poslovnih procesov v določenem podjetju je najboljši mogoč scenarij, če se v podjetju uporablja ERP [30]. V tem primeru so dnevniški zapisi standardizirani in dostopni za vsa orodja, ki so med seboj povezana. Za večino manjših podjetij, ki uporabljajo agilne metodologije, velja, da je ERP zaradi drage prilagodljivosti neprimeren [30]. Neredko se zgodi, da taka podjetja za vsak posamezen sklop procesov uporabljajo posebno orodje. Ker so v tem delu predmet študij primerov manjša podjetja z raznovrstnim naborom orodij, je treba njihovo število omejiti. Omejiti se je treba na orodja, ki celovito zajamejo določene vidike dela v podjetju. Ena od možnosti je osredotočanje zgolj na orodja, ki so tesno povezana s procesom razvoja programske kode. V tem primeru iz evalvacije izpustimo vse zaposlene, katerih delo s tem ni vsaj delno povezano.

3.2 Evalvacija

Cilj drugega dela je oceniti kakovosti razvojnih procesov, ki smo jih prepoznali pri načrtovanju. Osnovo za drugi del predstavlja pristop [68], ki ocenjuje

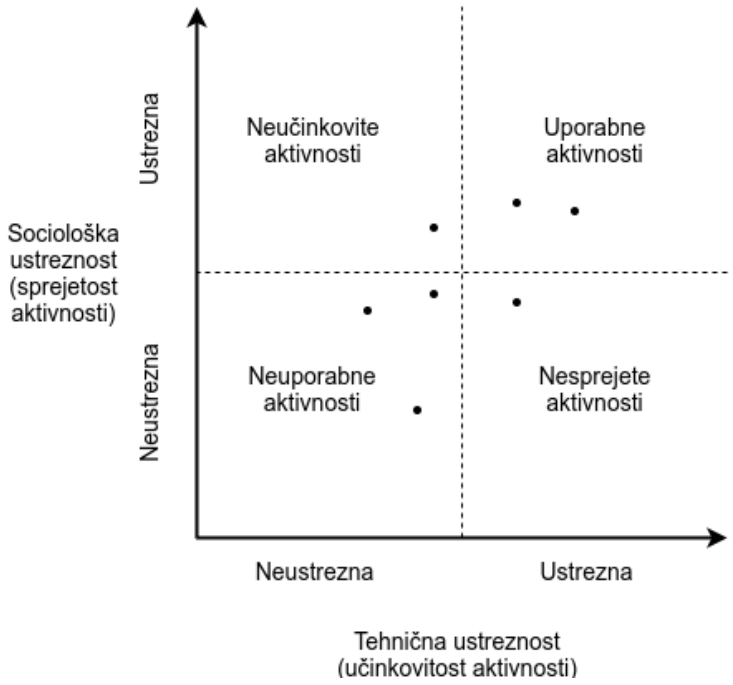
sociološki, tehnični in ekonomski vidik razvojnih procesov. Pri načrtovanju evalvacije smo kot ključne deležnike prepoznali razvijalce, tehnično napredno osebje in vodstvo, vsak je zadolžen za ocenjevanje določenega vidika. Sociološki vidik ocenjujejo razvijalci, tehnični vidik je ocenjen s strani tehnično naprednega osebja, ekonomski vidik pa ocenjuje vodstvo podjetja. Za vsak vidik imamo podane nabore karakteristik za ocenjevanje [67, 68], ki so predstavljeni v poglavju 2.1. Iz omenjenih naborov je treba izbrati podmnožico karakteristik, ki so najbolj primerni za obravnavano podjetje. Kljub temu da je izbira odvisna od ocenjevanih aktivnosti in podjetja, v katerem opravljamo raziskavo, smo v tem poglavju opisali in utemeljili izbiro za našo študijo primera. Razlog za to je, da želimo bralcu skozi primer olajšati razumevanje postopka izbire karakteristik.

Deležniki skozi ankete podajo ocene za izbrane karakteristike, ki so vezane na posamezno aktivnost ali orodje. Možni odgovori so oblikovani po sedemstopenjski Likertovi lestvici, prikazani na sliki 3.3. Lestvica meri kako zelo se anketiranec strinja s podano izjavo.

Slika 3.3: Likertova lestvica - možni odgovori anketiranca na podane izjave.

Ko pridobimo ocene aktivnosti iz vseh vidikov jih ustrezno razvrstimo v ogrodje evalvacijskega modela. Ogrodje lahko vizualiziramo z razsevnim diagramom (slika 3.4). Vsaka aktivnost je postavljena v določen kvadrant glede na oceno sociološkega in tehničnega vidika. Osi x in y predstavljata tehnično in sociološko ustreznost.

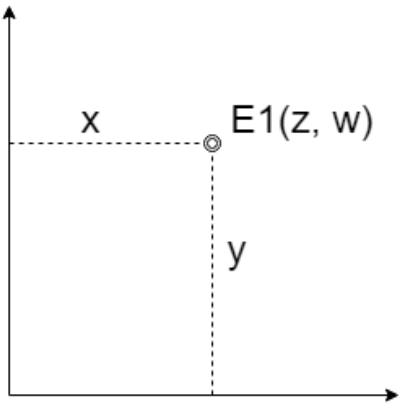
Poleg ocen tehničnega in sociološkega vidika sta predstavljena ekonomski vidik ter razpršenost rezultatov sociološkega vidika. Razpršenost nam pove, kako enotni so razvijalci pri svojih ocenah, česar iz povprečne ocene ne moremo razbrati. Za merjenje razpršenosti smo izbrali standardni odklon, ki velja za uveljavljeno mero razpršenosti na intervalu [8]. Za potrebe



Slika 3.4: Ogrodje evalvacijskega modela za aktivnosti, ki jih spremljamo.

našega pristopa smo predpostavili intervalnost Likertovih lestvic [69], saj je to že uveljavljena praksa med raziskovalci in nam omogoča računanje povprečij ter standardnih odklonov [28]. V našem pristopu smo uporabili merilo razpršenosti za prepoznavanje pojavov, ko se stopnja sprejetosti med raziskovalci močno razlikuje. Na aktivnosti, pri katerih je razpršenost največja, želimo opozoriti vodstvo podjetja. Tudi, če relativno majhno število zaposlenih slabo sprejme določeno aktivnost, lahko to negativno vpliva na uspešnost poslovnih procesov.

Primer vizualizacije ocene posameznega elementa je prikazan na sliki 3.5. E1 je v tem primeru oznaka elementa, razdalja x predstavlja povprečno oceno tehničnega vidika, razdalja y povprečno oceno sociološkega vidika, spremenljivka z je povprečna ocena ekonomskega vidika, w pa predstavlja razpršenost ocen sociološkega vidika.



Slika 3.5: Vizualizacija ocene posameznega elementa.

3.2.1 Sociološki vidik

Sociološki vidik meri stopnjo sprejetosti elementov in razloge za trenutno stopnjo sprejetosti. Iz nabora karakteristik [67] smo po posvetovanju s tehnično naprednim osebjem izbrali pet karakteristik oziroma dejavnikov, ki jih spremljamo. V tabeli 3.1 se nahaja predloga za vprašalnik o sociološkem vidiku aktivnosti. Elementi, ki jih ocenjujemo, se nanašajo na izvajanje določene aktivnosti, med katere spada tudi uporaba orodij. Za ocenjevanje trenutnega stanja sprejetosti izbranega elementa lahko ocenjujemo pogostost uporabe, če se za to pojavi priložnost, in konsistentnost uporabe. Pri konsistentnosti uporabe je treba upoštevati stopnjo zrelosti podjetij. Ker so predmet naše študije primera manjša in mlada podjetja obstaja možnost pomanjkanja natančnih navodil za izvajanje določenih aktivnosti [44]. Za merjenje konsistentnosti so potrebna natančno definirana navodila ali pravila (poglavje 2.1), zato smo to karakteristiko izpustili. Pogostost uporabe smo izbrali kot edini kazalnik trenutnega stanja socialne sprejetosti. Oceno te karakteristike lahko preverimo z uporabo dnevniških zapisov v primeru, da vsebujejo sledi izvajanja ocenjevane aktivnosti.

Naslednje štiri karakteristike temeljijo na subjektivnih odgovorih in jih ne moremo potrditi z uporabo dnevniških zapisov. Njihov namen je poiskati razloge za trenutno stanje sprejetosti in preveriti, zakaj se določena aktivnost

Pogostost uporabe	<Aktivnost/Orodje> izvajam oziroma uporabljam vedno, ko se za to pojavi priložnost
Relativna prednost	Uporabljanje oziroma izvajanje <aktivnosti/orodja> izboljša kakovost mojega dela
Socialna združljivost	Uporaba oziroma izvajanje <aktivnosti/orodja> je združljiva z načinom razvoja, ki ga prakticiram
Prostovoljnost	<Aktivnost/Orodje> izvajam prostovoljno
Dostopnost znanja	Imam dovolj priložnosti za pridobivanje znanja na področju izvajanja/uporabe <aktivnosti/orodja>

Tabela 3.1: Ogrodje vprašalnika, ki ocenjuje sociološki vidik

ne uporablja vsakič, ko se za to pojavi priložnost. Za posamezno aktivnost želimo vedeti, ali na pogostost uporabe vpliva zaznavanje izboljšanja kakovosti dela, združljivost z uveljavljenim načinom razvoja, prostovoljnost izvajanja ter dostopnost znanja. Možni razlogi za trenutno stanje so v naši študiji primera izbrani po posvetovanju s tehnično naprednim osebjem in vodstvi podjetij, ki so poudarila najverjetnejše razloge za nesprejetost procesov v njihovem podjetju.

3.2.2 Tehnični vidik

Tehnični vidik ocenjuje tehnično ustreznost oziroma učinkovitost izvajanja izbranih aktivnosti. Pri načrtovanju evalvacije je bilo izbrano tehnično napredno osebje, ki ocenjuje ta vidik. Po posvetovanju s tehnično naprednim osebjem smo za našo študijo primera izbrali pet karakteristik. Prva karakteristika, ki jo merimo, je pogostost priložnosti za izvajanje določene aktiv-

nosti. Omenjena karakteristika meri trenutno stopnjo tehnične učinkovitosti. Čeprav lahko razvijalci ocenijo da izvajajo aktivnost vsakič, je lahko priložnosti premalo, da bi bil element učinkovit. Naslednji štirje dejavniki, ki jih spremljamo pri tehničnem vidiku, nam pojasnijo razloge za trenutno stanje. Karakteristike vzrokov za trenutno učinkovitost, ki jih bomo merili, so ustreznost za razvojno ekipo, ustreznost za projekt, prilagodljivost razvijalcem ter skladnost z modernimi razvojnimi pristopi. Predloga za vprašalnik o tehničnem vidiku aktivnosti je prikazana v tabeli 3.2.

Priložnosti za uporabo	Priložnost za uporabo/izvajanje <aktivnosti/orodja> se pojavi pogosto.
Ustreznost za razvojno ekipo	Razvojna ekipa učinkovito izvaja/uporablja <aktivnost/orodje> .
Ustreznost za projekt	Izvajanje/uporaba <aktivnosti/orodja> ustreza tehničnim potrebam trenutnega projekta.
Skladnost z modernimi razvojnimi standardi	<Aktivnost/Orodje> je v skladu z modernimi razvojnimi standardi in pristopi
Prilagodljivost razvijalcem	<Aktivnost/Orodje> je možno prilagoditi različnim stopnjam znanja in izkušenosti razvijalcev.

Tabela 3.2: Ogrodje vprašalnika, ki ocenjuje tehnični vidik

Iz skupine atributov tehnične ustreznosti, opisane v poglavju 2.1, smo izpustili le ustreznost za stranke. Tehnično napredno osebje je namreč potrdilo, da izvajanje določenih aktivnosti sicer vpliva na stranke, a je težko oceniti kako močno. Poleg karakteristik tehnične ustreznosti smo izbrali tudi prilagodljivosti razvijalcem [14], saj se znanja in izkušnje razvijalcev znotraj podjetja razlikujejo.

3.2.3 Ekonomski vidik

Ekonomski vidik ocenjuje ekonomsko učinkovitost elementov. Za ocenjevanje tega vidika smo izbrali tri karakteristike: vpliv na izdelek, stroške in strateške cilje. Predloga za vprašalnik o ekonomskem vidiku je predstavljena v tabeli 3.3. Karakteristiko stroškov in vpliva na izdelek smo povzeli po pristopu, ki je bil uspešno testiran na majhnih podjetjih [68]. Omenjeni pristop ocenjuje tudi karakteristiko ciljev organizacije. Atkinson [6] razširi cilje organizacije v dve skupini. Prva skupina so cilji same organizacije, druga pa cilji deležnikov, med katere sodijo stranke, zaposleni, poslovni partnerji ipd. Vodstva podjetij morajo pri ocenjevanju ciljev organizacije upoštevati tudi vse prisotne deležnike. Razlogov za to je več. V manjših zagonskih podjetjih je pogosto prisoten pojav, da so zaposleni težje pogrešljivi kot pri podjetjih, ki imajo že uveljavljen izdelek [44, 14]. Vzroki za to so lahko nestandardiziranost procesov in pomanjkanje dokumentacije, ki bi omogočala prenos znanja. Zato je ena izmed ključnih funkcij vodstva tudi skrb za zadovoljstvo zaposlenih [14].

Vpliv na izdelek	Izvajanje/uporaba <aktivnost/orodja> poveča kakovost izdelka
Stroški	Izvajanje/uporaba <aktivnost/orodja> zmanjša stroške razvoja
Strateški cilj	Izvajanje/uporaba <aktivnost/orodja> pomaga podjetju doseči zastavljene strateške cilje

Tabela 3.3: Ogrodje vprašalnika, ki ocenjuje ekonomski vidik

Pri ocenjevanju strateških ciljev podjetja mora vodstvo upoštevati tudi morebitne negativne vplive aktivnosti. Lahko se namreč zgodi, da določen proces poveča stroške in ne vpliva na izdelek, a izboljša zadovoljstvo razvijalcev ali pa poveča zaupanje poslovnih partnerjev [29]. Primer slednjega je uvedba varnostnih standardov in protokolov, kar je pogosto drag proces, ampak lahko izboljša ugled podjetja [29, 18] in poveča zaupanje strank ali

partnerjev.

3.3 Analiza dnevniških zapisov

Z analizo dnevniških zapisov orodij za nadzor različic lahko pridobimo podatke, ki nam omogočijo vpogled v dejansko izvajanje razvojnih procesov. Pri podjetjih, ki se ukvarjajo z razvojem programske opreme, lahko te podatke pridobimo s pomočjo orodij za nadzor izvorne kode, ki spadajo v podmnožico VCS. Zaradi lažje berljivosti bomo izraz VCS v prihodnjih poglavjih omejili na orodja za nadzor izvorne kode. Pri vseh podjetjih, ki so predmet naše študije primera, kot VCS uporabljajo orodje Git.

Z neomejenim dostopom do podatkov, shranjenih z VCS, bi lahko teoretično merili produktivnost zaposlenih [21], napredovanje v znanju, konsistentnost, upoštevanje metodologije in ostale vidike dela razvijalcev, ki se nanašajo na izvajanje razvojnih procesov. Merjenje produktivnosti zaposlenih poleg merjenja izvajanja razvojnih procesov vključuje tudi merjenje številnih drugih aktivnosti [9]. Zato je lahko merjenje produktivnosti zgolj na podlagi ocene izvajanja razvojnih procesov zavajajoče. Ocenjevanje produktivnosti je zelo kompleksno, saj zahteva upoštevanje vseh aktivnosti posameznika v organizaciji [9]. Pri razvoju programske opreme je težko najti preprosta in uporabna merila za dnevniške zapise, ki bi omogočala natančno ocenjevanje produktivnosti vseh razvijalcev [21]. Merila, ki jih bomo predstavili v tem poglavju, niso namenjena merjenju produktivnosti zaposlenih, temveč merjenju sprejetosti razvojnih procesov. Glavna razlika je torej v predmetu ocenjevanja, kar mora biti jasno tudi vodstvom podjetij, v katerih se evalvacija izvaja.

Preden začnemo z izbiro meril, je treba upoštevati dostopnost dnevniških zapisov. Bolj kot so dnevniški zapisi podrobni, težje je pridobiti privolitev vodstva podjetja za dostop do njih. V orodju Git smo prepoznali tri različne stopnje podrobnosti dnevniških zapisov. Višja kot je stopnja, več podatkov vsebuje. Prepoznane stopnje so naslednje:

- **Prva stopnja** vsebuje samo podatek, da je določen avtor spremenil izvorno kodo, nekaj metapodatkov ter avtorjev opis spremembe.
- **Druga stopnja** vsebuje vse podatke iz prve stopnje, obogatene s seznamom datotek, ki so bile spremenjene. Za posamezno datoteko je podano tudi število izbrisanih in dodanih vrstic.
- **Tretja stopnja** vsebuje podatke prvih dveh stopenj, zaporedno številko ter vsebino dodanih in izbrisanih vrstic.

S tretjo stopnjo podrobnosti dnevniških zapisov je mogoča rekonstrukcija izdelka in celotne zgodovine njegovega razvoja. Zaradi varnostnih razlogov dostopa tretje stopnje nismo dobili. Dostop druge stopnje smo dobili v enem podjetju, drugi dve podjetji sicer nista omogočili dostopa do dnevniških zapisov, sta pa zato dovolili dostop do anonimiziranih rezultatov analize zapisov druge stopnje. Za izvedbo analize smo razvili program, ki spremlja izbrana merila (predstavljena v poglavju 3.3.3). Program uporablja samo metode statistične analize (računanje pogostosti, trendov ipd.). Podrobneje je opisan v poglavju 3.4. Rudarjenje procesov smo izvajali z orodjem Prom, ki poleg statistične analize uporablja tudi druge metode podatkovnega rudarjenja [65]. Zaradi prej omenjene omejitve dostopa smo lahko orodje Prom uporabili samo na dnevniških zapisih enega podjetja.

3.3.1 Razvojni procesi v orodju Git

Za izbiro ustreznih meril je treba razumeti delovanje izbranega orodja in poslovne procese, ki jih podpira. Za našo študijo primera smo se osredotočili na razumevanje delovanja orodja Git in na aktivnosti, ki jih ta podpira. Git je porazdeljeni VCS. Porazdeljeni VCS so tisti, pri katerih ima vsak razvijalec celotno kopijo zgodovine sprememb oziroma različic [1]. Razvijalec lahko poljubno ureja svojo kopijo zgodovine sprememb. Če želi deliti svoje spremembe z drugimi jih shrani v oddaljeno skladišče, ki je skupno celotni razvojni ekipi. Pri centraliziranih VCS razvijalci nimajo svoje kopije zgodo-

vine sprememb. Vsako spremembo morajo shraniti v oddaljeno skladišče, na katerem se nahaja edina skupna zgodovina sprememb [10].

Uporaba orodja Git se lahko med podjetji razlikuje, zato smo se osredotočili na postopke, ki jih uporabljajo opazovana podjetja. Vsak postopek je sestavljen iz ene ali več aktivnosti, naštetih v nadaljevanju.

- **Inicializacija orodja** - na začetku se lahko razvijalec odloči, ali želi naložiti že obstoječo zgodovino sprememb iz oddaljenega skladišča (ukaz *clone*), ali želi ustvariti novo zgodovino sprememb (ukaz *init*). V obeh primerih se na izbranem mestu ustvari t. i. izvorni direktorij. Orodje Git spremlja vse spremembe v izbranih datotekah, ki se nahajajo znotraj izvornega direktorija. Razvijalec lahko določi katere datoteke naj sistem spremlja (ukaz *add*).

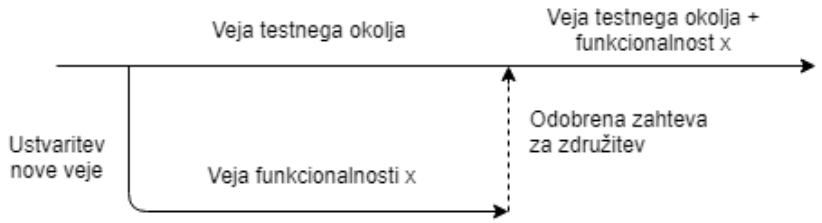
- **Ustvarjanje ločene veje za novo funkcionalnost** - vsak razvijalec naj bi ustvaril ločeno vejo (ukaz *branch*) za novo funkcionalnost, ki jo želi razviti. Veja je posebna različica zgodovine trenutne kode. Z delom na ločenih vejah se razvijalci izognejo morebitnim konfliktom. Do konfliktov lahko pride, če spremenjena koda prvega razvijalca spremeni delovanje na novo spremenjene kode drugega razvijalca, ali če razvijalci spreminjajo isti del kode.

- **Shranjevanje izvorne kode** - ko je razvijalec zadovoljen s trenutno kodo oziroma ko razvije neko funkcionalnost, lahko shrani spremembe (ukaz *commit*). Pri tem mora podati sporočilo, ki opisuje novo funkcionalnost. Spremembe se shranijo lokalno, na trenutno izbrano vejo.

- **Shranjevanje sprememb v oddaljeno skladišče** - lokalne spremembe je treba shraniti v oddaljeno skladišče (z ukazom *push*), do katerega imajo dostop drugi razvijalci v skupini. Takoj ko je sprememba postavljena, postane vidna drugim razvijalcem, ki jo lahko sinhronizirajo s svojo lokalno različico kode. Spremembe se shranijo na izbrano vejo na oddaljenem skladišču. Pri shranjevanju v oddaljeno skladišče

lahko pride do konfliktov, če drugi razvijalec na isto vejo shrani svojo različico.

- **Združevanje vej** - ko je določena funkcionalnost dokončana, razvijalec ustvari zahtevo za združitev veje, na kateri se ta funkcionalnost nahaja, z osrednjo vejo, ki vsebuje kodo, ki naj bi bila pripravljena za postavitev na produkcijsko oziroma testno okolje. Razvijalec, ki je zadolžen za strokovne preglede, preveri ustreznost kode, ki je označena za združitev in jo odobri ali zavrne. Primer združevanja je prikazan na sliki 3.6



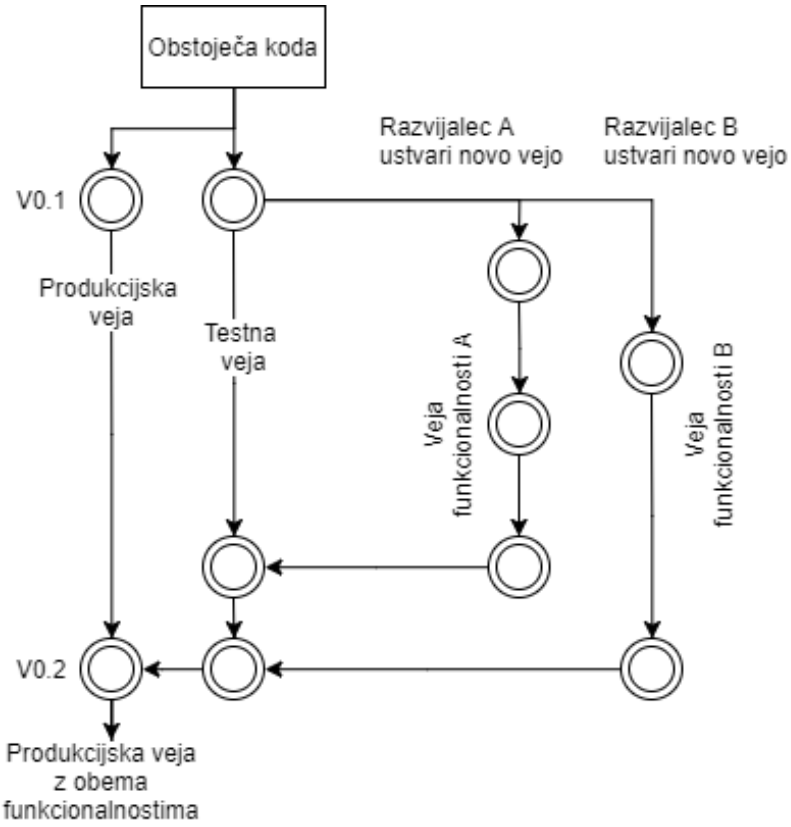
Slika 3.6: Prikaz vejitve in združevanja izvorne kode.

- **Popravljanje trenutne različice** - v primeru, da je bila napaka v kodi odkrita, preden je koda shranjena v oddaljeno skladišče, je treba popraviti trenutno različico (ukaz *reset*). S tem želimo zagotoviti, da je vsaka različica kode v oddaljenem skladišču delujoča.

- **Označevanje produkcijskih različic** - testirano in delujočo kodo na produkcijski veji se po navadi označi s številčnimi ali kakšnimi drugimi smiselnimi oznakami. Označevanje produkcijskih različic pripomore k spremljanju in vidljivosti napredka.

Primer postopka ustvarjanja novih funkcionalnosti z orodjem Git je prikazan na sliki 3.7. Vsak razvijalec ustvari lastno lokalno kopijo kode. Vsak krog v diagramu predstavlja shranjevanje novih sprememb kode. Razvijalci testirajo kodo, preden jo shranjujejo. Ko je enkrat funkcionalnost dokončana, pošljejo zahtevo po združitvi na testno vejo. V primeru odobritve združevanja se delovanje preveri tudi v testnem okolju. Po tem je funkcionalnost pripravljena za

produkcijsko okolje. Vsaki spremembi produkcijskega okolja je treba dodati lastno oznako (V0.1 in V0.2 v primeru na sliki 3.7).



Slika 3.7: Primer celotnega postopka ločevanja vej in shranjevanja različic pri razvoju novih funkcionalnosti.

3.3.2 Struktura dnevniških zapisov

Orodje Git ima možnost izpisa dnevniških zapisov po meri. Izpis prilagodimo glede na podatke, ki jih želimo vključiti v analizo. V podjetjih, ki so predmet študije primera, nam je zaradi zaupnosti omogočen dostop le do številčnih podatkov, ki povzemajo spremembe v datotekah. Kljub omenjeni omejitvi pa lahko nad navedenimi podatki opravimo vrsto meritev, ki nam omogočajo boljše razumevanje dejanskega izvajanja razvojnega procesa. Podatki, ki smo jih vključili za potrebe naše študije primera, so naslednji:

- Avtor kode in avtor različice
- Datum spremembe
- Opis spremembe
- Imena datotek, ki so bile spremenjene
- Število izbrisanih ali dodanih vrstic za vsako spremenjeno datoteko.

Nato definiramo obliko izpisa dnevniških zapisov. Pri tem moramo upoštevati vse podatke, ki jih želimo vključiti. Izpis dnevniških zapisov smo prilagodili z naslednjim ukazom:

```
git log --pretty=format:
"start%n%h;%ae;%at;%cE;%ct;%d;%n
startcomment%n%s;%nend" --numstat
> project_gitlog.log
```

Poleg uporabljenega ukaza *log* obstaja tudi ukaz *reflog*, ki vsebuje več informacij. Prednost ukaza *reflog* je, da vsebuje tudi celotno zgodovino Git ukazov, ki jih je razvijalec uporabljal. To pomeni, da lahko deterministično rekonstruiramo celotno zgodovino njegove interakcije z orodjem Git. Glavna pomanjkljivost tega izpisa je, da se izpišejo zgolj lokalno shranjeni ukazi. To pomeni, da bi za sestavitev celotnega procesa verzioniranja kode morali

imeti dostop do lokalne kopije dnevniških zapisov vseh razvijalcev. Zaradi te omejitve, smo ostali pri izpisu ukaza *log*.

Primer enega vnosa iz pridobljenih dnevniških zapisov je na sliki 3.8. Vsaka nova različica (shranjevanje kode) se začne z besedo *start*, sledi identifikacijski niz vnosa, avtor funkcionalnosti, čas shranjevanja funkcionalnosti, avtor vnosa, čas vnosa, kazalec veje, ki kaže na ta vnos. Nato je podano sporočilo vnosa, obdano s ključnimi besedami. Na koncu zapisa je prisoten še seznam, sestavljen iz vnosov, ki vsebujejo število dodanih vrstic, število izbrisanih vrstic ter ime in pot do datoteke, na katero se te dve številki nanašata. Na sliki 3.8 lahko zasledimo spremembe števila vrstic v sedmih datotek.

```
start
59b34fdd;anonuser12;1495373426;anonuser12;1495373426;; (HEAD -> develop, origin/develop);
startcomment
Updated user preference logic.;
end
12      4       api/maps/serializers.py
1       5       api/users/serializers.py
6       1       api/locations/serializers.py
5       33      api/urls.py
3       1       api/users/views.py
1       3       apps/users/admin.py
12      5       apps/users/tests.py
```

Slika 3.8: Prilagojen izpis orodja Git.

Dnevniški izpisi v taki obliki še vedno niso primerni za rudarjenje procesov z orodjem Prom, ki kot vhod sprejme samo format XES. Preoblikovanje podatkov pa ni tako trivialno, saj je treba upoštevati njihov pomen (v ogrodju CRISP, opisanem v poglavju 2.2.1, je stopnja razumevanja podatkov pred pripravo podatkov).

3.3.3 Pregled in izbira ustreznih meril

Postopek pregleda in izbire ustreznih meril spada v prvi del našega pristopa (načrtovanje, kot je prikazano na sliki 3.1), vendar je zaradi konteksta dnevniških zapisov opisan v tem poglavju. V tem koraku se je treba posvetovati s tehnično naprednim osebjem, ki mora izbrati katera merila so

zanje najbolj pomembna. Merila smo razdelili v tri skupine, glede na njihovo kompleksnost. V nadaljevanju so naštete in opisane vse tri skupine meril.

- **Prva skupina meril** - merijo aktivnosti, katerih izvajanje lahko neposredno razberemo iz dnevniških zapisov orodja. Primeri takih aktivnosti za orodje Git so opisani v poglavju 3.3.1. Za prvo skupino je značilno, da se pri vsaki novi izvedbi aktivnosti zabeleži nov dnevniški vnos. Zabeleženi dnevniški vnos nedvoumno odraža izvedbo določene aktivnosti.

- **Druga skupina meril** - merijo izvajanje bolj kompleksnih aktivnosti. Izvajanja teh aktivnosti niso neposredno razvidna iz dnevniških zapisov. Za definiranje meril moramo bolj podrobno razumeti vsebino dnevniškega zapisa in na podlagi vsebine določiti, katero aktivnost odraža.

- **Tretja skupina meril** - merijo izvajanje aktivnosti, katerih sledi lahko prepoznamo s preučevanjem odvisnosti dveh ali več dnevniških zapisov. V to skupino spadajo merila, ki so sestavljena iz poljubnega zaporedja meril prve in druge skupine.

Primeri meril, ki smo jih definirali za uporabo v naši študiji primera, so predstavljeni v tabelah 3.4 in 3.5. Za vsako merilo smo definirali identifikacijsko oznako (stolpec *Id*), opis merila (stolpec *Merilo*), katero stopnjo dostopa potrebujemo (stolpec *dost.*), stopnjo kompleksnosti merila (stolpec *kompl.*) in katero aktivnost odraža (stolpec *Aktivnosti*). V nadaljevanju se bomo na primere meril sklicevali po njihovih identifikacijskih oznakah.

Za pridobivanje meritev druge skupine se moramo poglobiti v tehnično plat dnevniških zapisov. V to skupino spadajo dnevniški zapisi orodja Git, ki vsebujejo številčne informacije o spremembah v datotekah repozitorija. Zato je treba definirati, katero aktivnost odraža sprememba določene datoteke. Preslikavo iz datoteke v aktivnost imenujemo abstrakcija na nivoju dnevniških zapisov [52]. Glavni izziv je razumevanje različnih tipov datotek in kakšno vlogo imajo v projektu. Podatki, ki jih imamo v dnevniških

Id	Merilo	dost.	kompl.	Aktivnost
M1	Št. dnevniških vnosov	1	1	Shranjevanje različic z orodjem Git
M2	Št. združitev različnih vej	1	1	Ustvarjanje nove veje za novo funkcionalnost
M3	Št. prekratkih opisov	1	1	Pisanje jedrnatih opisov različic
M4	Št. ključnih besed v opisih	1	2	
M5	Št. oznak različic	1	1	Označevanje produkcijskih različic
M6	Št. sprememb vrstic v dokumentacijskih datotekah	2	2	Pisanje dokumentacije
M7	Št. omenjenih funkcij v dokumentaciji	3	3	
M8	Št. sprememb vrstic v testnih datotekah	2	2	Pisanje testov
M9	Št. testnih funkcij	3	2	
M10	Št. novih vrstic v kodi	2	2	Pisanje novih funkcionalnosti
M11	Št. novih funkcij v kodi	3	2	
M12	Št. sprememb vrstic konfiguracije	2	2	Konfiguriranje projekta

Tabela 3.4: V tabeli je predstavljen prvi del definiranih meril s pripadajočimi aktivnostmi.

M13	Št. majhnih različic	3	2	Odpravljanje hroščev
M14	Št. oznak v opisih verzij	1	1	
M15	Št. izbrisanih vrstic	1	2	Preurejanje kode
M16	Št. ali delež zamenjanih vrstic	3	2	
M17	Št. sprememb testov / Št. sprememb kode	2	2	Testno vodeni razvoj
M18	Št. novih funkcij v testih / Št. novih funkcij	3	3	
M19	Št. majhnih zaporednih popravkov	2,3	3	Testiranje pred shranjevanjem
M20	Št. nedelujočih / Št. vseh različic	3	3	
M21	Št. oznak produkcijskih različic	1	1	Uporaba oznak za produkcijske različice

Tabela 3.5: V tabeli je predstavljen drugi del definiranih meril s pripadajočimi aktivnostmi.

zapisih, so lokacija, ime in končnica datoteke, ki označuje njen tip. Za primer lahko vzamemo sliko 3.8. Če razvijalec piše v datoteko, ki vsebuje teste, je to aktivnost pisanja testov, podobno velja za datoteke, ki vsebujejo dokumentacijo (aktivnost pisanja dokumentacije), konfiguracijo (aktivnost konfiguriranja projekta), izvorno kodo (aktivnost pisanja novih funkcionalnosti) ipd. Primer abstrakcij, upoštevan tudi v našem pristopu, je prikazan v tabeli 3.6. Vsako spremembo v datoteki določene vrste klasificiramo kot izvajanje določene aktivnosti. V prvem stolpcu so opisane aktivnosti, v drugem so klasifikacijske oznake, v zadnjem so predstavljene končnice, ki določajo vrsto datotek.

Aktivnost	Oznaka klasifikacije	tipične lastnosti datotek
Pisanje novih funkcionalnosti	CODE	končnice .java, .py, .cpp, .c, ipd.
Pisanje testov	TEST	končnice .test, .spec; niz *test* v imenu, lokaciji datoteke, ipd.
Konfiguracija	CONF	končnice .conf, niz *conf* v imenu, lokaciji datoteke, ipd.
Pisanje dokumentacije	DOC	končnice .doc, .md, .pdf, ipd.

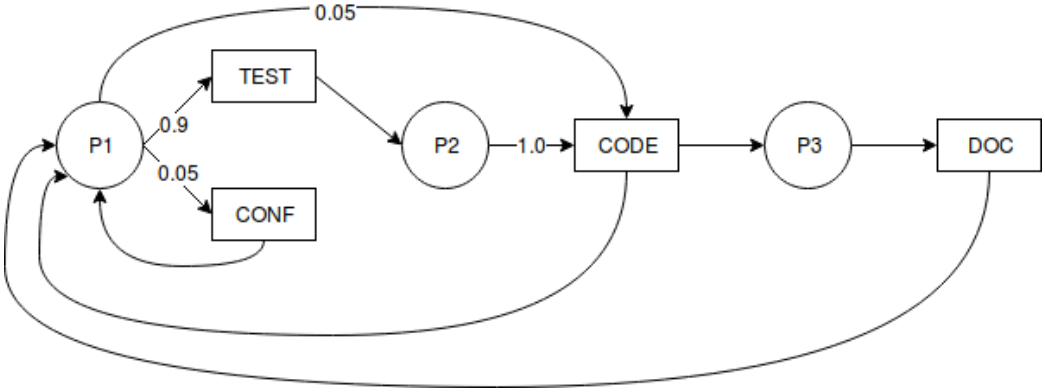
Tabela 3.6: V tabeli so predstavljeni štirje primeri preslikav sprememb datotek v aktivnosti.

Izvorna koda projektov je po navadi omejena na en programski jezik, vendar prisotnost več različnih ne vpliva na uspešnost analize. Pomembno je zgolj prepoznati tip datoteke iz končnice, kar ni težko, saj vsak programski jezik po navadi uporablja le eno končnico. Problem prepoznavanja datotek izvorne kode je torej trivialen. Za večino ostalih datotek (konfiguracija, testiranje, dokumentacija ipd.) je prepoznavanje namena težje, saj je veliko več

svobode in možnosti pri določanju lokacij, imen in končnic tovrstnih datotek. Zaradi takih primerov je potrebna prisotnost tehnično naprednega osebja, ki pozna projekte in lahko nedvoumno določi želene preslikave. Izvajanje vsake izmed aktivnosti lahko najpreprosteje preverimo z izračunom pogostosti izvajanja (primer so merila M6, M8, M10 in M12). Po tem kriteriju se ne razlikujejo bistveno od prve skupine aktivnosti (aktivnosti neposredno povezane z orodjem Git - M1-M5).

Za pridobivanje meritev tretje skupine meril moramo preučiti povezanost dnevniških zapisov. Najbolj očitni primeri takih aktivnosti (poudarjeni s strani tehnično naprednega osebja) so izvajanje testno vodenega razvoja (angl. *Test Driven Development* - TDD), obvezno pisanje dokumentacije za določene module in izvajanje testov pred shranjevanjem kode. Najbolj preprosto je preverjanje izvajanja testno vodenega razvoja. Cilj te aktivnosti je povečati kakovost izdelka z uvedbo obveznega pisanja testov pred dejansko kodo [7]. Testi so neposredno izpeljani iz zahtev, nato je pa koda posodobljena do te mere, da se novi testi izvedejo uspešno. Tak pristop razvijalce navadi na konsistentno pisanje testov in zmanjša možnost razvoja kode, ki ne bi ustrezala podanim zahtevam. Omenjeno aktivnost je možno spremljati tako s statistično analizo kot z rudarjenjem procesov, pomembno je, da se aktivnost pisanja testov vedno pojavi v paru s samim razvojem kode. Iz ene shranjene različice kode ne moremo ugotoviti točnega zaporedja, ali so bili prej napisani testi ali koda, saj je časovni žig za obe aktivnosti enak. Kljub tej pomanjkljivosti lahko spremljamo primere, ko razvijalci ne upoštevajo pristopa TDD. Tak primer lahko hitro ugotovimo, če določena različica kode vsebuje zgolj aktivnosti pisanja kode, ne pa tudi testov. Podoben postopek je tudi pri obveznem pisanju dokumentacije za določene module, le da tu spremljamo drugi tip aktivnosti in to le v primerih, ko se aktivnost pisanja kode izvaja na datoteki, ki pripada ustreznemu modulu. Za definiranje zaporedij aktivnosti lahko uporabimo Petrijeve mreže, ki se pogosto uporabljajo za modeliranje, analizo in kontrolo diskretnih dogodkovnih sistemov [25]. Primer vizualizacije zaporedja aktivnosti s pomočjo Petrijeve mreže je na

sliki 3.9. Slika prikazuje primer, ko se razvijalci v dobri meri držijo pristopa



Slika 3.9: Primer Petrijeve mreže z verjetnostmi za posamezne prehode.

TDD. Kot vidimo, lahko razvijalec izbere eno izmed treh aktivnosti, vendar je pričakovati, da bo najpogosteje začel s testi. To je v opazovanih podjetjih razumljivo, saj se večina dela sestoji iz posodabljanja in dodajanja novih funkcionalnosti, za kar je po pristopu TDD potrebno urejanje ali dodajanje testov. Za nekatere primere testi niso potrebni, prav tako naj ne bi bili potrebni za spreminjanje konfiguracije. Pri razvoju ali posodabljanju nekaterih komponent je obvezno pisanje dokumentacije (oznaka *DOC*).

Petrijeve mreže lahko dobimo kot rezultat rudarjenja procesov z orodjem Prom [65]. Pri podjetjih, kjer imamo prost dostop do podatkov, smo uporabili program, ki smo ga razvili za potrebe te naloge in orodje Prom. Statistična analiza nam služi kot osnova za potrjevanje izvajanja določenih aktivnosti, pridobili pa smo jo s programom, ki smo ga razvili. Rudarjenje procesov z orodjem Prom smo uporabili kot pomožen postopek, za odkrivanje zaporedij aktivnosti pri razvoju funkcionalnosti. Za ta postopek je treba poleg klasificiranja aktivnosti poznati tudi kontekst podatkov [52]. Na sliki 3.8 sta vidni dve aktivnosti (oznaki *TEST*, *CODE*), vendar v različnih kontekstih, prvi kontekst so spremembe na komponenti *api* s poudarkom na razvijalcih, drugi pa spremembe zaledne logike (komponenta *apps*). Spremembe na različnih komponentah moramo obravnavati v različnih primerih in ustvariti posebno Petrijevo mrežo. Pri ugotavljanju konteksta posame-

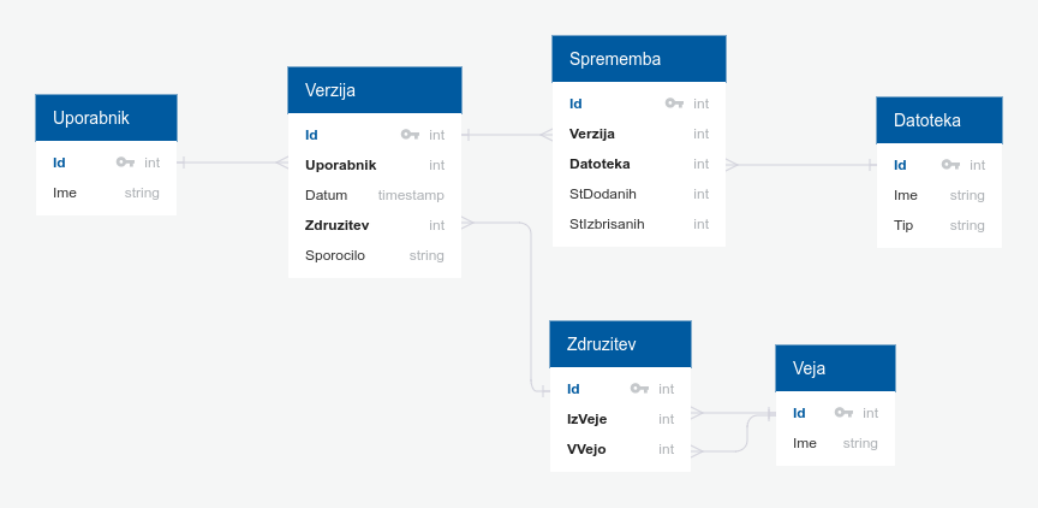
zne datoteke smo se morali posvetovati z razvijalci in tehnično naprednim osebjem.

3.3.4 Priprava podatkov za analizo

Ko zberemo dovolj informacij za razumevanje podatkov sledi postopek priprave oziroma preoblikovanja dnevniških zapisov. Vhod za ta korak predstavlja dnevniški zapis predstavljen na sliki 3.8. Obstaja veliko orodij, ki so namenjena pridobivanju različnih statističnih podatkov iz dnevniških zapisov orodja Git, vendar so večinoma omejena na aktivnosti, ki so z njim neposredno povezane (primeri teh aktivnosti so opisani v poglavju 3.3.1). V našem primeru je bilo treba izdelati orodje po meri [45], saj smo poleg omenjenih tipičnih meril vključili tudi dodatna merila, specifična za posamezna podjetja.

Najprej smo se lotili preoblikovanja dnevniških zapisov v obliko, ki bi omogočila čim lažjo statistično obdelavo podatkov. Prvi korak tega postopka je izdelava ustreznega podatkovnega modela. Vsak dnevniški zapis smo razčlenili na informacije o avtorju, različici, združevanju vej in spremembah v datotekah. Vsak dnevniški zapis ima torej enega avtorja, eno različico, ki vsebuje časovno komponento, nič ali eno združitev vej, ter toliko sprememb, kolikor je datotek. Podatkovni model, ki smo ga izdelali, je prikazan na sliki 3.10. Razčlenjeni podatki so shranjeni kot objekti znotraj programa.

Preoblikovani podatki nam omogočajo preproste poizvedbe glede na predmet raziskovanja. Če želimo dobiti na primer število vseh različnih vej, lahko preprosto preštejemo število objektov tipa *Veja*. Merila, ki jih bomo spremljali, so podrobneje opisana v poglavju 3.4. Uporaba podatkovnega modela nam omogoča tudi preprosto predelavo dnevniških zapisov v format, ki je potreben za orodje rudarjenja procesov. Vsak objekt tipa *Sprememba* je en vnos v novem seznamu dnevniških zapisov. Dodan jim je podatek o imenu razvijalca, datumu spremembe in imenu ter tipu datoteke, iz katerih bo izpeljana preslikava. Po potrebi so na podoben način dodani tudi vnosi združitve vej.



Slika 3.10: Podatkovni model, ki omogoča preprosto pridobivanje različnih statističnih podatkov s poizvedbami.

Na koncu je treba odstraniti podvojene vnose. Primer rezultata je prikazan v tabeli 3.7. Izhod je shranjen v standardnem formatu XES (opisanem v poglavju 2.2.3) in pripravljen za uporabo.

Orodje	Proces	Id primera	Aktivnost	Časovni žig	Izvajalec
Git	Razvoj programske opreme	API	CODE	1495373426	anonuser12
Git	Razvoj programske opreme	APPS	TEST	1495373426	anonuser12
Git	Razvoj programske opreme	APPS	CODE	1495373426	anonuser12

Tabela 3.7: Primer preoblikovanja dnevniških zapisov prikazanih na sliki 3.8.

3.4 Potrjevanje in poročanje rezultatov

Cilj analize dnevniških zapisov je omogočiti tehnično naprednemu osebju vpogled v merljive podatke o izvajanju aktivnosti. Poudarek je na preverjanju pogostosti izvajanja določenih aktivnosti. Na podlagi teh meritev bo tehnično osebje podalo svojo oceno o pogostosti izvajanja aktivnosti. Ker je splošno optimalno pogostost izvajanja nemogoče določiti [21], mora tehnično napredno osebje imeti oblikovana določena pričakovanja in kriterije, na podlagi katerih bodo interpretirali meritve. Možnosti preverjanja izvajanja posameznih aktivnosti z dnevniškimi zapisi so različne. Spodaj so naštete najbolj pogoste aktivnosti, ki jih je tehnično napredno osebje poudarilo, in opisana merila, ki jim lahko pomagajo pri določanju ocene pogostosti izvajanja. V oklepaju so podane lastnosti, na katere vpliva izvajanje izbrane aktivnosti.

- **Shranjevanje kode z orodjem Git** (dostopnost kode) - za večji del razvoja je ta postopek nujen, saj je edini način za deljenje sprememb izvorne kode znotraj podjetja. Pojavijo se lahko izjeme, ko razvijalci razvijejo pomožna orodja, ki jih iz različnih razlogov ne shranjujejo z orodjem Git. Merjenje pogostosti te aktivnosti je trivialno (M1), saj vsak dnevniški zapis vsebuje podatke o avtorju in časovnem žigu.

- **Ustvarjanje nove veje za novo funkcionalnost** (modularnost kode, urejenost zgodovine) - namen uporabe je opisan v poglavju 3.3.1. Tako kot pri shranjevanju kode, lahko tudi tukaj merimo pogostost ustvarjanja novih vej posameznih razvijalcev (M2) in število primerjamo s številom novih funkcionalnosti, ki so bile ustvarjene v tem časovnem okvirju. Merilo lahko vpliva na oceno pogostosti uporabe. V primeru, da razvijalci uporabljajo ukaz *rebase* za združevanje vej, stare veje pa izbrišejo, je to merilo neuporabno, ker se v zgodovino ne zapiše podatek o združevanju vej.

- **Pisanje testov** (testiranje) - pri testiranju lahko primerjamo deleže napisanih testov z deležem celotne kode posameznega razvijalca. Merilo

deleža testov v kombinaciji s številom vrstic (M8) nam lahko da okvirno sliko o tem, ali razvijalci pišejo teste. Primerjanje števila vrstic pa je lahko zavajajoče, saj je odvisno od učinkovitosti programiranja - pri neizkušenih razvijalcih lahko pride do podvojevanja kode. V primeru dostopnejših zapisov namesto števila vrstic uporabimo število funkcij ali drugih komponent v testih (M9)

- **Pisanje dokumentacije** (dokumentacija) - pri aktivnosti dokumentiranja lahko merimo pogostost spreminjanja datotek (M6), ki vsebujejo dokumentacijo. Omenjeno merilo je ustrezno v primeru, da se dokumentacija shranjuje z orodjem Git. Za preverjanje konsistentnosti pisanja dokumentacije lahko uporabimo tudi orodje Prom za rudarjenje procesov. V tem primeru preverjamo pogostost prehodov na aktivnost pisanja dokumentacije v modulih, kjer je ta aktivnost zahtevana. V primeru dostopnejših zapisov bi lahko spremljali število ali delež v dokumentaciji omenjenih komponent iz kode (M7).

- **Pisanje novih funkcionalnosti in posodobitev** (delo na izdelku) - pisanje izvorne kode, v to kategorijo spada razvoj novih funkcionalnosti. Za to aktivnost lahko merimo število novih vrstic v določenem časovnem okvirju (M10). Bolj natančno merilo (M11) bi lahko uporabili v primeru dostopnejših dnevniških zapisov, saj bi lahko merili število novih funkcij in ostalih komponent v kodi.

- **Konfiguriranje projekta** (delo na izdelku) - pri konfiguriranju projekta lahko spremljamo pogostost sprememb v datotekah, ki vsebujejo konfiguracijo (M12).

- **Preurejanje kode in popravki** (delo na izdelku) - menjava neustrezne kode s primernejšo (angl. *refactoring*). Za merjenje izvajanje te aktivnosti lahko spremljamo število izbrisanih vrstic v določenem časovnem obdobju (M15). Z dostopom druge stopnje ne moremo vedeti, ali je bila določena vrstica zamenjana ali samo izbrisana. V pri-

meru dostopnejših dnevniških zapisov bi lahko na podlagi te informacije merili to aktivnost (M16).

- **Upoštevanje testno vodenega razvoja** (konsistentnost testiranja) - v primeru, da se dodajajo nove funkcionalnosti (M6), želimo preverjati prisotnost sprememb v testnih datotekah. Merimo torej delež različic, kjer je prišlo do pisanja testov hkrati s pisanjem novih funkcionalnosti (M17, M18). Ustreznost zaporedij lahko merimo tudi z orodjem Prom.

- **Pisanje jedrnatih opisov različic** (urejenost zgodovine) - preverjamo dolžino in število besed pri shranjenih različicah. Pri meritvah se osredotočimo predvsem na neustrezne izvedbe aktivnosti, torej primere kjer so opisi prekratki ali nesmiselni. V primeru, da imajo razvijalci dobro definirano logiko označevanja različic bi lahko opise uporabili za merjenje pogostosti izvajanja številnih drugih aktivnosti (M14).

- **Uporaba oznak za produkcijske različice** (urejenost zgodovine) - preverjamo prisotnost *tag* vnosov v projektu (M21).

- **Izvajanje strokovnih recenzij s pomočjo zahtevkov *pull* (*merge*)** (kakovost kode) - izvajanja te aktivnosti ne moremo potrditi, lahko jo pa zavrnemo, v primeru, da ne najdemo nobene sledi o prisotnosti drugih vej. Za merilo uporabimo kar število vej (M2).

- **Testiranje pred shranjevanjem kode** (kakovost kode) - z dostopom druge stopnje izvajanja te aktivnosti ne moremo potrditi, lahko jo delno zavrnemo s preverjanjem števila in periode shranjevanj različic. Če se za različico z velikim številom sprememb hitro pojavi več različic z manjšimi spremembami na istih datotekah (M19), lahko sklepamo da je to posledica popravljanja napak, ki jih je vsebovala nova funkcionalnost in so nastali zaradi pomanjkanja testiranja. V primeru dostopnejših zapisov bi lahko izvajali preverjanje pravilnosti kode za vsako različico in delež nepravilnih različic podali kot merilo (M20).

Iz definiranih meril je razvidno, da pogosto ne moremo natančno določiti kako zelo se razvijalci držijo posameznih aktivnosti. Večinoma nam merila pomagajo ugotoviti, kateri razvijalci se popolnoma izogibajo izvajanju določene aktivnosti ali pa to zelo redko počnejo. Glede na to, da lahko tehnično napredno osebje poda oceno o pogostosti priložnosti za uporabo posameznih elementov, smo predpostavili, da zna tudi iz podanih meritev prepoznati, ali razvijalci izvajajo določeno aktivnost vsakič, ko se pojavi priložnost. To velja v primeru, da prepoznajo podane meritve kot uporabne. Če razvijalci ocenijo, da aktivnost izvajajo vsakič, zapisi pa so zelo redki z majhnim številom sprememb kljub pogostim priložnostim za uporabo, mora tehnično osebje podati novo oceno pogostosti, ki upošteva rezultate analize dnevniških zapisov. Poleg trenutnega stanja bo tehnično naprednemu osebju podana tudi informacija o trendu izvajanja aktivnosti, če so na voljo dnevniški zapisi za daljše časovno obdobje. Glede na to, da ima tehnično osebje poleg skupnih meritev o izvajanju aktivnosti na voljo tudi meritve za posameznega razvijalca, lahko na podlagi analize dnevniških zapisov poda tudi oceno enotnosti razvijalcev glede na pogostost izvajanja aktivnosti.

Posamezne karakteristike ocenjevanja meril so ocenjene z vrednostmi Likertove lestvice. Ogrodje vprašalnika je prikazano v tabeli 3.8. Tehnično

Uporabnost meritev	Rezultati meritev so uporabni za oceno pogostosti izvajanja/uporabe <aktivnosti/orodja>
Ocena pogostosti uporabe/izvajanja na podlagi meritev	<Aktivnost/Orodje> se izvaja/uporablja vsakič ko se za to pojavi priložnost
Ocena razpršenosti pogostosti uporabe/izvajanja na podlagi meritev	Razvijalci so glede na pogostost uporabe/izvajanja <aktivnosti/orodja> enotni

Tabela 3.8: Ogrodje vprašalnika za upoštevanje rezultatov analize dnevniških zapisov.

osebje najprej oceni uporabnost meritev. V primeru, da je ocena meritev negativna ali nevtralna (manjša ali enaka 4) se odgovarjanje na naslednji dve vprašanji izpusti, saj neuporabnih meritev ne želimo upoštevati. Če tehnično osebje oceni meritev kot uporabno, lahko poda oceno pogostosti izvajanja na podlagi meritev. V primeru, da tehnično osebje meni, da so razvijalci ustrezno ocenili pogostost izvajanja lahko ocenjevanje na podlagi dnevniških zapisov izpustijo. Končna ocena sociološkega vidika je izračunana kot povprečje ocen vseh karakteristik, predstavljenih v tabeli 3.1, pri čemer je ocena pogostosti izvajanja izračunana kot povprečje ocene, ki jo podajo razvijalci na podlagi lastnega zaznavanja in ocene, ki jo na podlagi dnevniških zapisov poda tehnično osebje. To pomeni, da enakovredno upoštevamo ocene, pridobljene na podlagi zaznavanja razvijalcev in tiste, ki so pridobljene na podlagi dnevniških zapisov. Za tak postopek smo se odločili zaradi pomanjkanja podobnih pristopov, ki bi jih lahko uporabili kot referenčne primere. Ocene razvijalcev smo se odločili upoštevati kot enakovredne, ker so se izkazale kot koristne v obstoječih raziskavah, ki uporabljajo zgolj ocene zaznavanja razvijalcev [67, 68]. V omenjenih raziskavah se je tudi tehnično osebje izkazalo kot sposobno oceniti število priložnosti za uporabo, kar je pogoj za uspešno interpretacijo rezultatov dnevniških zapisov.

Na koncu lahko tehnično osebje poda tudi oceno enotnosti razvijalcev pri pogostosti izvajanja aktivnosti. V primeru, da se pogostosti izvajanja med razvijalci zelo razlikujejo, je treba ugotoviti razloge za tako stanje in o tem obvestiti vodstvo podjetja. Vloga razpršenosti je opisana v poglavju 3.2.

Poglavje 4

Študija primera

Razviti pristop smo preizkusili v praksi skladno z metodo študije primera [73]. Na praktičnih primerih smo želeli preizkusiti razviti pristop in primerjati rezultate z in brez upoštevanja dnevniških zapisov in nato ugotoviti, kakšna je bila dodana vrednost.

V študiji primera smo s kvantitativnimi in kvalitativnimi metodami ocenili sprejetost in učinkovitost razvojnih procesov z uporabo razvitega evalvacijskega pristopa. Kvantitativne metode študije primera v našem pristopu vključujejo anketiranje zaposlenih, statistične metode analize rezultatov anketiranja in dnevniških zapisov ter rudarjenje procesov. Med kvalitativne metode spadajo pogovori s tehnično naprednim osebjem in vodstvom podjetja, ki so nam omogočili prilagoditev pristopa izbranim podjetjem in interpretacijo kvantitativnih podatkov. Izvedli smo pluralno študijo primera (angl. *multiple case study*) v treh podjetjih. Pri pluralni študiji primera moramo upoštevati logiko replikacije. Ta služi kot podlaga za posploševanje rezultatov za vse podobne primere. Glede na podobnost primerov ločimo dobesedno in teoretično replikacijo [73]. V primeru dobesedne replikacije napovemo podobne rezultate v vseh primerih, pri teoretični replikaciji pa napovemo različne rezultate pri čemer moramo podati razloge za napovedano različnost.

Za testiranje pristopa smo izbrali tri primere podjetij in se osredotočili

na podobno velike razvojne ekipe. Zaradi podobnosti razvojnih procesov v vseh treh podjetjih smo se odločili za dobesedno replikacijo. To pomeni, da pri vseh treh podjetjih pričakujemo podobne rezultate pri ocenjevanju uporabnosti dnevniških zapisov za evalvacijo razvojnih procesov. S testiranjem razvitega pristopa v izbranih podjetjih smo ocenili uporabnost analize dnevniških zapisov in tudi koristnost celotnega pristopa, ki temelji na omenjeni analizi.

4.1 Opis podjetij

Študijo primera smo izvedli v treh podjetjih, katerih primarna dejavnost je razvoj programske opreme. V vsakem podjetju je zaposlenih več kot deset ljudi, večji del zaposlenih so razvijalci. Vsa tri podjetja razvijajo programske rešitve, ki jih uporablja veliko število končnih strank. Načrtovanje in razvoj se prilagajata na podlagi povratnih informacij s trga. Čeprav se pristopi k trženju izdelkov razlikujejo, so procesi razvoja programske opreme zelo podobni. Podobnost osnovnega poteka dela razvijalcev nam je omogočila spremljanje enakih ali podobnih aktivnosti. Za podobne aktivnosti smo v vseh treh podjetjih uporabili enaka merila pri analizi dnevniških zapisov. Spremljanje enakih meril nam je omogočilo dobesedno replikacijo rezultatov ocenjevanja uporabnosti teh meril.

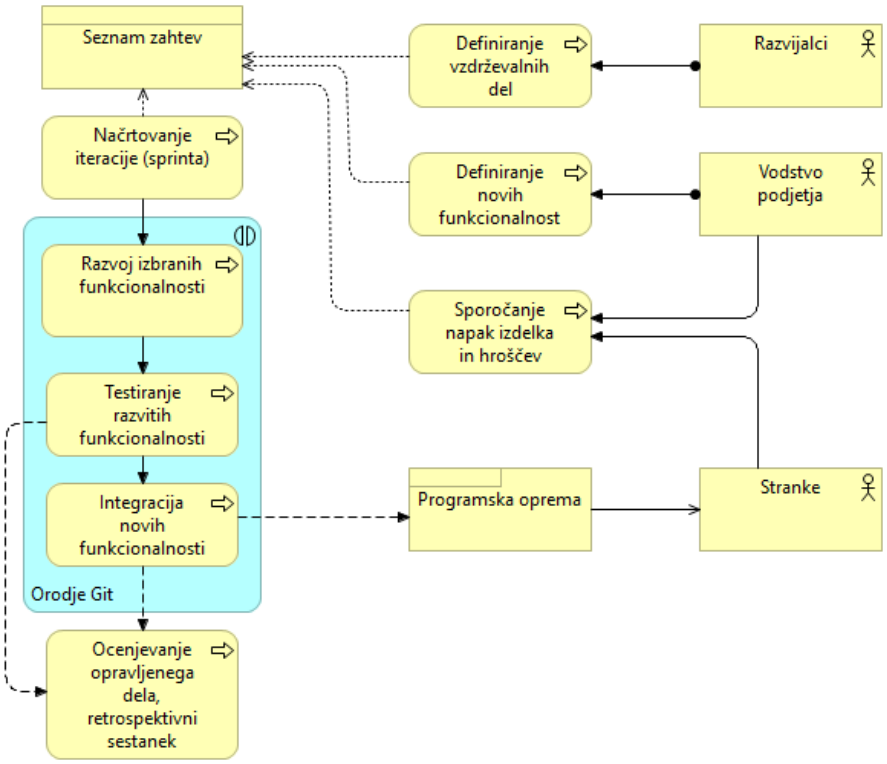
Večina zaposlenih v izbranih podjetjih so razvijalci programske opreme, zato smo v ospredje postavili aktivnosti in orodja, ki jih uporabljajo razvijalci. Podjetja za svoje poslovne procese uporabljajo različna specializirana orodja. Analiza dnevniških zapisov vsakega uporabljenega orodja bi pripomogla k boljšemu razumevanju poteka poslovnih procesov, vendar bi hkrati povečala kompleksnost in časovno zahtevnost evalvacije. Povečana časovna zahtevnost je predstavljala oviro za vodstva podjetij, saj zahteva večjo vključenost tehnično naprednega osebja pri povezovanju dnevniških zapisov in definiranju meril. Zato se je bilo treba osredotočiti na eno orodje, katerega dnevniški zapisi bi odražali čim večje število razvojnih procesov.

Pri načrtovanju evalvacije (poglavje 3.1) smo ugotovili, da vsa tri podjetja uporabljajo orodje Git za nadzor izvorne kode. Izkazalo se je, da je Git edino orodje v izbranih podjetjih, ki vsebuje zapise o zgradbi izdelka in njegovih spremembah skozi čas. Izbira dnevniških zapisov orodja Git je bila zato edina smiselna možnost. Dnevniški zapisi orodja Git nam omogočajo vpogled v sam proces razvoja programske kode, posredno pa lahko dobimo tudi informacije o testiranju funkcionalnosti, planiranju dela, obsegu in vrsti funkcionalnosti ter postavljanju izdelka na produkcijo. Dobra stran izbire orodja Git za spremljanje aktivnosti je podobnost postopka uporabe tudi v različnih podjetjih. To pomeni, da lahko večino izbranih meril nespremenjeno uporabimo v več primerih. Uporaba enakih meril ne pomeni nujno tudi uporabe enakih kriterijev za interpretacijo meritev. Norme pri izvajanju aktivnosti se med podjetji razlikujejo. Zahtevana pogostost testiranja v enem podjetju je lahko bistveno manjša kot v drugih podjetjih. V naši študiji smo pričakovali, da bo ocena uporabnosti posameznega merila podobna v vseh treh primerih.

4.1.1 Podjetje A

Evalvacijo smo najprej izvedli v podjetju, ki ponuja storitve sledenja in nadzora vozil. Podjetje je staro tri leta in ima več tisoč strank. Ima trinajst zaposlenih, od tega je devet razvijalcev. V podjetju smo najprej prepoznali vodstvo in tehnično napredne zaposlene. S pomočjo slednjih smo prepoznali vloge zaposlenih v podjetju. Vloge razvijalcev se delijo na razvijalce zalednega dela sistema, čelnega dela sistema in strojne programske opreme. Kljub razvoju programske opreme za različne platforme so številne aktivnosti enake. Osrednji proces razvoja, ki velja za vse razvijalce, je prikazan na sliki 4.1. V podjetju se uporablja vitki razvojni pristop Scrum [33]. Razvoj je razdeljen na časovne iteracije, ena iteracija oziroma sprint traja približno deset delovnih dni. V stopnji načrtovanja se razvijalcem določijo naloge, ki jih morajo izpolniti do konca sprinta. Tipične naloge so razvoj novih funkcionalnosti, odpravljanje napak v izdelku in vzdrževalna dela. V večini

primerov nove funkcionalnosti zahteva vodstvo podjetja, odpravljanje napak v izdelku zahtevajo uporabniki izdelka, vzdrževalna dela pa izvirajo iz omejitev uporabljenih tehnologij in jih razvijalci sami definirajo. V odvisnosti od kompleksnosti funkcionalnosti lahko njihov razvoj traja tudi več sprintov. Ko je funkcionalnost razvita, testirana in odobrena s strani vodstva, jo razvijalci integrirajo z izdelkom.

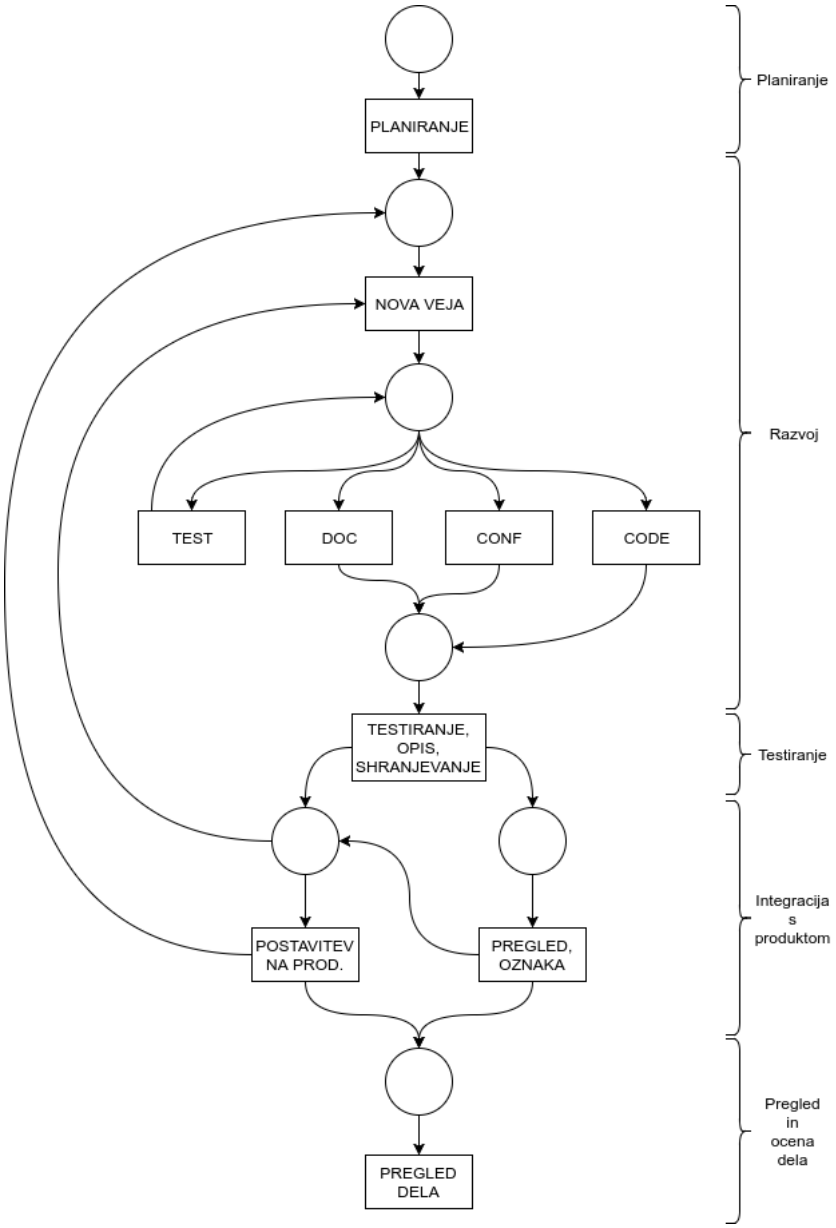


Slika 4.1: Elementi ene iteracije postopka razvoja programske opreme v podjetju A.

Metodologija določa uporabo določenega nabora orodij in aktivnosti. V procesu načrtovanja se pretežno uporablja orodje Trello [59], ki vsebuje seznam in opis zahtev, nalog, trenutno stanje nalog in zadolžene razvijalce. Za komunikacijo med razvijalci se uporablja orodje Slack [55], ter različni odjemalci elektronske pošte. V razvojne procese je najbolj vpeto orodje Git, ki ga uporabljajo vsi razvijalci. Vodstvo tega podjetja nam je omogočilo do-

stop do dnevniških zapisov vseh projektov vendar z omejitvijo druge stopnje (stopnje so opisane v poglavju 3.3). Načrtovanje evalvacije smo zaključili z izbiro meril, ki je opisana v poglavju 4.1.4 za vsa tri podjetja hkrati.

Zaradi neposrednega dostopa do dnevniških zapisov smo imeli možnost uporabe orodja Prom za rudarjenje procesov. Omenjeno orodje smo uporabili za ustvaritev Petrijeve mreže z verjetnostmi za posamezne prehode. Iz rezultatov smo želeli razbrati zaporedja aktivnosti in verjetnosti za posamezne postopke. Postopek iz slike 4.1 smo razčlenili na manjše aktivnosti in ga na sliki 4.2 vizualizirali s Petrijevo mrežo. Prikazana Petrijeva mreža odraža pravilna zaporedja izvajanja aktivnosti. Za aktivnosti konfiguriranja, pisanje dokumentacije, testov in kode smo uporabili oznake, opisane v poglavju 3.3.3. Kot rezultat rudarjenja procesov smo pričakovali podobno Petrijevo mrežo z nekaj omejitvami. Prva omejitev je bila pomanjkanje podatkov o nekaterih aktivnostih. Iz dnevniških zapisov nam ni uspelo pridobiti podatkov o izvajanju vseh želenih aktivnostih, zato smo se osredotočili na zaporedja aktivnosti razvoja in shranjevanja (označena na sliki 4.2). V tem podjetju smo pričakovali rezultate, ki odražajo izvajanje testno vodenega razvoja in dokumentiranja. V rezultatih smo pričakovali, da so zaporedja, pri katerih je testiranje pred pisanjem kode in ki vključujejo dokumentiranje, v večini.



Slika 4.2: Petrijeva mreža razčlenjenega osrednjega postopka razvoja programske opreme v podjetju A.

4.1.2 Podjetje B

Drugo podjetje je zagonsko podjetje, staro štiri leta. Ima deset zaposlenih, od tega je sedem razvijalcev. Njihov glavni izdelek je platforma za spletno

trženje dogodkov. Stranke so različni organizatorji dogodkov, ciljne stranke pa so vsi, ki se zanimajo za udeležbo na teh dogodkih. V podjetju je zaposlenih enajst ljudi, od tega je sedem razvijalcev. Prepoznali smo tudi tehnično naprednega razvijalca in vodjo. Vloge razvijalcev se tu delijo na razvijalce zalednih sistemov in razvijalce čelnega dela sistema. Tudi v tem podjetju se uporablja Scrum za planiranje in izvedbo razvoja. Večina aktivnosti razvijalcev je neodvisnih od platforme, za katero razvijajo programsko opremo. Potek osrednjega postopka razvoja programske opreme je podoben kot pri podjetju A.

Za nadzor izvorne kode se uporablja orodje Git. V tem podjetju nismo dobili neposrednega dostopa do dnevniških zapisov. Dobili smo dovoljenje izvedbe programa za analizo dnevniških zapisov in uporabo rezultatov tega programa. Izvajanje analize je potekalo kontrolirano na napravah v lasti podjetja B. Omejitev neposrednega dostopa do dnevniških zapisov nam je onemogočilo izvedbo rudarjenja procesov z orodjem Prom. Analiza z omenjenim orodjem je namreč preveč časovno zahtevna za izvedbo na gostujočih napravah, zahteva preoblikovanje dnevniških zapisov in omogoča njihov pregled, kar ni bilo v interesu podjetja. Kljub naštetim omejitvam smo lahko še zmeraj pridobili meritve za večino aktivnosti.

4.1.3 Podjetje C

Podjetje C je srednje veliko podjetje z več kot petdesetimi zaposlenimi. Več kot polovica zaposlenih so razvijalci. Podjetje je staro približno deset let. V podjetju se ukvarjajo s širokim spektrom različnih projektov, ki so večinoma usmerjeni v področje računalništva in informatike. Zaradi omenjene raznolikosti smo se odločili poiskati večji oddelek, ki je po uporabljeni metodologiji najbolj homogen. Izbrali smo oddelek, ki se ukvarja z raziskovalnimi projekti za večje naročnike. Ko je izdelek končan, vzdrževanje ni več skrb tega oddelka, zato imajo malo opravka z dejanskimi uporabniki izdelka. Ukvarjajo se z razvojem zalednih in čelnih delov sistema. V tem oddelku smo identificirali šest razvijalcev, tehnično naprednega razvijalca ter vodjo. Za-

radi drugačnega poslovnega modela se tudi razvojni procesi bolj razlikujejo od ostalih dveh podjetij. Kljub temu je osnovni postopek razvoja podoben razvojnim ekipam iz podjetij A in B. Planiranje razvoja se izvaja v odvisnosti od vpletenosti naročnika v razvoj. V primeru, da ni posebnih zahtev naročnika, se uporablja prilagojena različica pristopa Scrum.

Tako kot v drugih dveh podjetjih se tudi tu uporablja Git za nadzor izvorne kode. Dostop do dnevniških zapisov je dovoljen z enakimi omejitvami kot pri podjetju B.

4.1.4 Izbira aktivnosti in meril za ocenjevanje

Po izvedbi prvih dveh korakov načrtovanja evalvacije (opisanih v poglavju 3.1) je bilo treba izbrati aktivnosti in merila za ocenjevanje. Ugotovili smo, da so razvojni procesi v izbranih podjetjih podobni. Ker smo se odločili za uporabo logike dobesedne replikacije, je bilo treba razvojne procese razčleniti na aktivnosti, ki so skupne vsem trem podjetjem. Spremljanje podobnih aktivnosti nam omogoča lažjo primerjavo rezultatov kakovosti posameznih meril. Za ocenjevanje smo se zato osredotočili na tipične aktivnosti podjetij, ki se ukvarjajo z razvojem programske opreme. To so testiranje, preurejanje, pisanje dokumentacije, testov, novih funkcionalnosti ipd. Del aktivnosti je enak pri vseh treh podjetjih tudi zaradi uporabe orodja Git.

Izbrane aktivnosti za vsa tri podjetja so prikazane v tabeli 4.1. V prvem stolpcu so definirane identifikacijske oznake za posamezne aktivnosti, drugi stolpec vsebuje imena aktivnosti, naslednji trije pa vsebujejo podatke o spremljanih aktivnostih in uporabljenih merilih v vseh treh podjetjih. Črka *T* označuje ali se določena aktivnost ocenjuje v podjetju. Oznake, ki se začnejo s črko *M* označujejo različna merila, pomen oznak pa je definiran v poglavju 4.1.4. Oznaka *RP* označuje uporabo orodja Prom za rudarjenje procesov za pridobivanje dodatnih meril pri ocenjevanju.

Čeprav smo pri definiranju aktivnosti poskušali izbrati čim bolj podobne, v nekaterih primerih to ni bilo možno. Prvi primer je aktivnost načrtovanja sprinta, ki velja za podjetji A in B, medtem ko ga v podjetju C nadomesti

Id	Aktivnost	Podj. A	Podj. B	Podj. C
A1	Shranjevanje različic z orodjem Git	T,M1	T,M1	T,M1
A2	Ustvarjanje nove veje za novo funkcionalnost	T,M2	T,M2	T,M2
A3	Načrtovanje sprinta	T	T	
A4	Zajem zahtev			T
A5	Pisanje jedrnatih opisov različic	T,M3,M4	T,M3	T,M3
A6	Pisanje testov	T,M8	T,M8	T,M8
A7	Pisanje dokumentacije	T,M6,RP	T	T,M6
A8	Pisanje novih funkcionalnosti	T,M10	T,M10	T,M10
A9	Konfiguriranje projekta	T,M12	T,M12	T,M12
A10	Preurejanje kode	T,M15	T,M15	T,M15
A11	Upoštevanje testno vodenega razvoja	T,M17,RP		
A12	Testiranje vseh funkcionalnosti		T,M17	
A13	Testiranje pred shranjevanjem kode	T,M19	T,M19	T,M19
A14	Uporaba oznak za produkcijske različice	T,M21	T,M21	T,M21
A15	Izvajanje strokovnih recenzij	T,M2	T,M2	T,M2
A16	Pregled dela v sprintu	T	T	
A17	Poročanje o napredku			T

Tabela 4.1: V tabeli so predstavljene aktivnosti vseh treh podjetij, ki jih bomo merili.

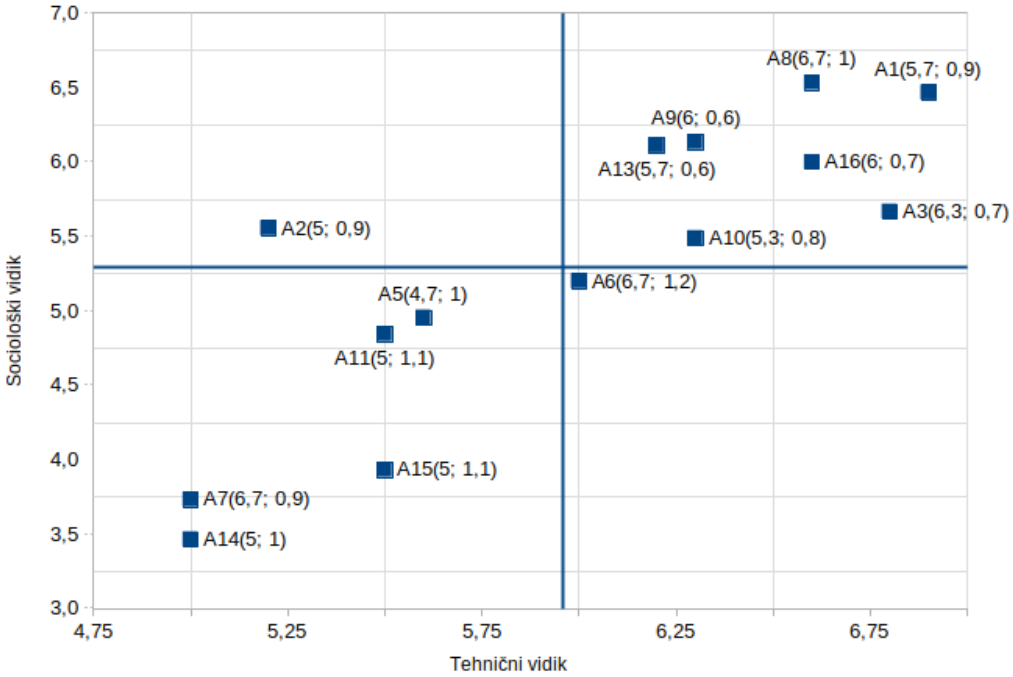
aktivnost *zajem zahtev*. Vzrok je ta, da se je v podjetju C treba prilagajati naročnikom projektov in ni klasičnega načrtovanja sprinta, tako kot pri podjetjih, ki sledijo metodologiji Scrum. Podobno je tudi z aktivnostjo poročanja o napredku, ki nadomesti pregled dela v sprintu. V podjetju C je bilo izpuščeno ocenjevanje aktivnosti A11 in A12. Obe aktivnosti sta povezani s pisanjem testov za vsako funkcionalnost. V izbrani ekipi te aktivnosti niso formalizirane zaradi pogostega prototipiranja, saj se lahko z njihovo uvedbo zmanjša produktivnost [7]. Podjetje A uporablja testno vodeni pristop pri razvoju programske opreme, medtem ko v podjetju B ni zahtevana njegova uporaba, je pa zahtevano testiranje vseh funkcionalnosti. Razlika je v vrstnem redu pisanja [7], v podjetju A se testi pišejo pred kodo, medtem ko v podjetju B to ni zahtevano. Z izbranimi aktivnostmi in merili za ocenjevanje, smo nadaljevali z izvedbo evalvacije po korakih opisanih v poglavju 3.2.

4.2 Rezultati evalvacij v podjetjih

V vseh treh podjetjih smo najprej pridobili povratno informacijo zaposlenih za potrebe ocenjevanja vseh treh vidikov. To smo storili s pomočjo anket, ki smo jih izdelali z orodjem Google Forms [24]. Vsak zaposleni je ocenjeval aktivnosti izbrane v poglavju 4.1.4 z ustreznim ogrodjem vprašalnika, kot je opisano v poglavju 3.2. Za vsako podjetje smo nato vizualizirali pridobljene podatke z razsevnim diagramom. Nesprejete in neučinkovite aktivnosti smo nato podrobneje obravnavali. Pri pregledu posamezne aktivnosti, ki smo jo izbrali za podrobnejšo obravnavo, smo tehnično naprednemu osebju priložili pripadajoče meritve. Na podlagi teh meritev so podali ocene tako, kot je opisano v poglavju 3.4. Za razumevanje vzrokov nesprejetih ali neučinkovitih elementov smo se dodatno posvetovali s tehnično naprednim osebjem. V primeru odstopanj med oceno pogostosti izvajanja, ki so jo podali razvijalci, in oceno, pridobljeno iz dnevniških zapisov, smo se posvetovali tudi o vzrokih za omenjeno razliko. Po določitvi končnih ocen smo rezultate posredovali vodstvu podjetja in se posvetovali o njihovi uporabnosti.

4.2.1 Podjetje A

Zaposleni podjetja A so ocenjevali 14 aktivnosti, ki so predstavljene v tabeli 4.1. Sociološki vidik je ocenjevalo devet razvijalcev, tehnični sta ocenjevala dva zaposlena, ekonomski pa samo direktor podjetja. Rezultati, vizualizirani po modelu opisanem v poglavju 3.2, so predstavljeni na sliki 4.3. Povprečne ocene karakteristik vseh treh vidikov so prikazane v tabeli 4.2.



Slika 4.3: Ocene sociološkega, tehničnega in ekonomskega vidika v podjetju A.

Iz rezultatov smo ugotovili, da nobena aktivnost ni ocenjena kot neustrezna iz tehničnega vidika, saj imajo vse oceno 5 ali več. Tri aktivnosti so bile ocenjene kot nesprejete (A7, A14, A15), ker imajo oceno sociološkega vidika manj kot 4. Želeli smo preveriti tudi zakaj so določene aktivnosti slabše ocenjene kot druge. Zato smo aktivnosti razdelili na bolje in slabše sprejete ter bolj in manj učinkovite glede na povprečje vseh ocen. V kategorijo manj učinkovitih in slabše sprejetih spadajo aktivnosti A5, A7, A11, A14 in A15. Med bolje sprejete a manj učinkovite spada aktivnost A2, med slabše sprejete

Sociološki vidik	Pogostost uporabe	Izboljša kakovost dela	Združljivo z delom	Prostovoljnost	Dostopnost znanja	Skupna ocena
A1	6,3	6,6	6,4	6,4	6,6	6,5
A2	5,2	5,4	4,9	6,0	6,2	5,6
A3	5,8	6,0	6,2	4,9	5,4	5,7
A5	5,0	3,4	4,9	6,1	5,3	5,0
A6	5,6	5,9	5,1	5,2	4,2	5,2
A7	3,9	3,8	3,6	3,9	3,6	3,7
A8	6,6	6,7	6,8	6,6	6,1	6,5
A9	6,1	6,3	6,0	6,1	6,1	6,1
A10	5,2	5,9	5,9	5,2	5,2	5,5
A11	4,9	5,0	5,1	4,9	4,3	4,8
A13	5,9	6,3	6,2	5,9	6,2	6,1
A14	2,7	2,3	3,1	2,7	6,6	3,5
A15	3,3	4,1	4,2	3,3	4,7	3,9
A16	5,8	6,1	6,1	5,8	6,2	6,0
Tehnični vidik	Pogostost priložnosti	Ustreznost za ekipo	Ustreznost za projekt	Moderni standardi	Prilagodljivost razvijalcem	Skupna ocena
A1	6,5	7	7	7	7	6,9
A2	6	5	5	6	4	5,2
A3	7	7	7	7	6	6,8
A5	7	6	6	7	2	5,6
A6	6,5	6	6,5	6	5	6
A7	5	5,5	6	5,5	3	5
A8	6,5	7	6	6,5	7	6,6
A9	5,5	7	6,5	7	5,5	6,3
A10	5,5	6	6	7	7	6,3
A11	6	5,5	6	6,5	3,5	5,5
A13	7	7	6,5	7	3,5	6,2
A14	5	5	6	7	2	5
A15	6	5	5,5	6	5	5,5
A16	7	7	7	6	6	6,6
Vodstveni vidik	Vpliv na produkt	Stroški	Strateški cilij			Skupna ocena
A1	5	6	6			5,7
A2	5	4	6			5,0
A3	7	5	7			6,3
A5	5	4	5			4,7
A6	7	6	7			6,7
A7	6	7	7			6,7
A8	7	6	7			6,7
A9	6	6	6			6,0
A10	6	3	7			5,3
A11	6	4	5			5,0
A13	6	5	6			5,7
A14	5	4	6			5,0
A15	6	3	6			5,0
A16	6	6	6			6,0

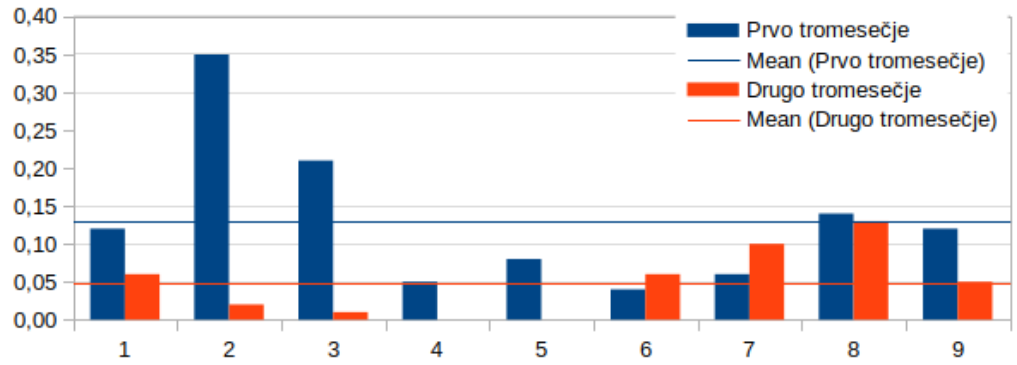
Tabela 4.2: Povprečne ocene posameznih vidikov in karakteristik v podjetju A.

in bolj učinkovite pa A6. Naštetim aktivnostim smo se podrobneje posvetili in preverili ustrezne meritve iz dnevniških zapisov. Za vsako izmed aktivnosti smo ocenjevali tudi razloge za trenutno stopnjo sprejetosti oziroma tehnične učinkovitosti. Razpršenost ocen bolje sprejetih in bolj učinkovitih aktivnosti

je povsod približno enaka 1 ali celo manjša. To pomeni, da so bile ocene sociološkega vidika dovolj enotne in aktivnosti nismo dodatno obravnavali. V nadaljevanju smo našteli vse obravnavane aktivnosti, začeli smo s tistimi, ki imajo najnižjo oceno. Pri interpretaciji rezultatov nam je pomagalo tehnično osebje.

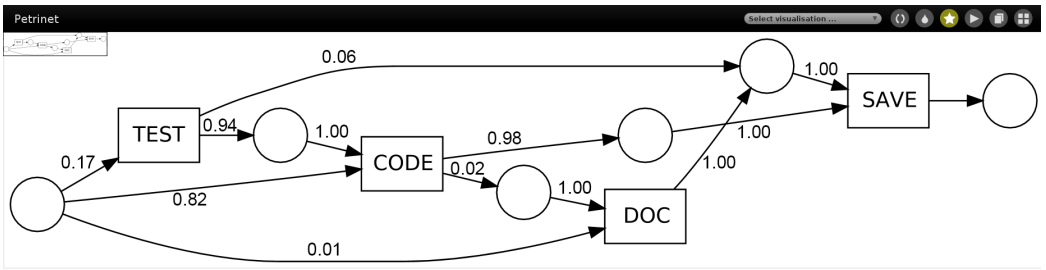
Pisanje dokumentacije (A7) se v podjetju izvaja na več različnih mestih v odvisnosti od njenega namena. Za dokumentiranje izvorne kode se dokumentacijske datoteke skupaj z izvorno kodo dodajajo v sistem za nadzor izvorne kode. Primer je modul, ki vsebuje kodo za programski vmesnik (angl. *Application Programming Interface* - API). Tam je dokumentacija ključnega pomena, saj omogoči razvijalcem razumevanje funkcij, brez da bi se poglabljali v razumevanje kode. V večini projektov tehnično osebje pričakuje pravočasno različico dokumentacije, ki odraža uporabo napisanih funkcij. Odsotnost dokumentacije lahko poveča čas, potreben za razumevanje funkcij, povzroči pa lahko tudi napake v kodi, ki nastanejo kot posledica nepričakovanega delovanja funkcije. Zaradi omenjenih razlogov ima aktivnost visoko oceno iz ekonomskega vidika. Ocena pogostosti uporabe, če se pojavi priložnost, je v območju nevtralnih številk (okoli 4). Omenjeno karakteristiko smo preverili tudi z uporabo merila števila dodanih vrstic (M6) in uporabo Petrijevih mrež (RP).

S programom, ki smo ga razvili, smo najprej pridobili podatke o napisani dokumentaciji vsakega posameznega razvijalca v zadnjih dveh tromesečjih. Uporabili smo merilo deleža dodanih vrstic v dokumentaciji v številu vseh dodanih vrstic. Rezultati deleža spreminjanja dokumentacijskih datotek za posameznega razvijalca so prikazani na sliki 4.4. Iz meritev je razvidno, da se je delež dokumentacije v drugem tromesečju zmanjšal. Preden smo se poglobili v razloge za zmanjšano aktivnost pisanja dokumentacije, smo želeli izničiti možnost, da je to posledica spreminjanja kode, za katero ni eksplicitno zahtevano pisanje dokumentacije. To smo storili z analizo dnevniških zapisov v modulu *api*, za katerega je zahtevano pisanje dokumentacije. Dnevniške zapise za modul *api* smo analizirali s programom Prom. Pridobljena Petri-



Slika 4.4: Delež sprememb dokumentacijskih datotek za vsakega razvijalca (Podjetje A, merilo M6).

jeva mreža je prikazana na sliki 4.5, uporabljene oznake pa so predstavljene v tabeli 3.6. Iz slike je razvidno, da je dokumentacija posodobljena le v

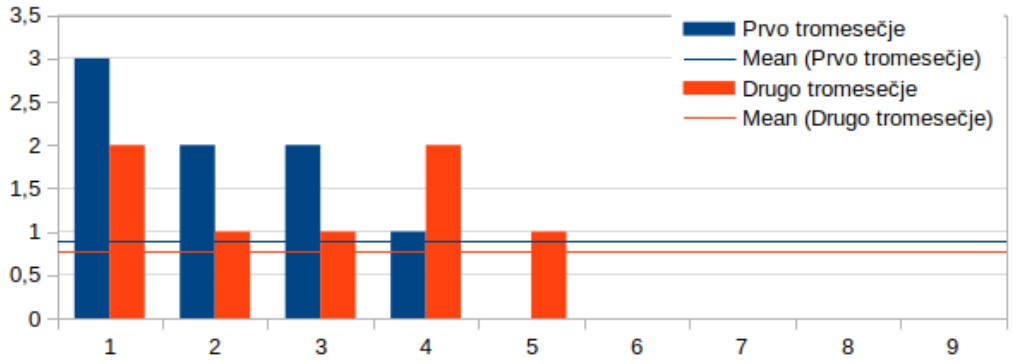


Slika 4.5: Vizualizacija Petrijeve mreže z verjetnostmi prehodov za modul *api* (Podjetje A, merilo RP).

slabih treh odstotkih primerov, kljub pogostim aktivnostim pisanja kode. Razvidno je tudi, da so uporabniki v slabih dveh odstotkih primerov dokumentacijo posodabljali skupaj s kodo v približno enem odstotku primerov pa samostojno. To pomeni, da razvijalci v stotih verzijah približno trikrat posodobijo dokumentacijo. Tehnično osebje je pričakovalo, da najmanj vsaka peta sprememba vsebuje posodobitev dokumentacije. V nasprotnem primeru se lahko zgodi, da dokumentacija ne vsebuje informacij o novih spremembah, ko jo razvijalci uporabljajo, kar lahko privede do napak pri razvoju. Poleg neažurnosti dokumentacije je za tehnično osebje neprimeren tudi njen obseg.

Pričakovali so namreč, da bo v drugem tromesečju večji delež dodane dokumentacije kot v prvem. Razlog za to je, da so v prvem tromesečju v ospredju bila vzdrževalna dela, v drugem pa so začeli z razširitvijo vmesnikov in razvojem novih funkcionalnosti. Na podlagi premajhnega obsega dokumentiranja (M6) in prenizke ažurnosti (RP) je tehnično osebje ocenilo, da je relativna pogostost izvajanja te aktivnosti manjša, kot so to ocenili razvijalci. Kot glavni vzrok za slabo sprejetost so navedli pomanjkanje pravil in smernic pri oblikovanju dokumentacije.

Označevanje produkcijskih različic (A14) se v podjetju uporablja za označevanje različic kode z oznako izdaje (angl. *release*). Z označevanjem se ohranja urejena zgodovino izdane kode, ki lahko olajša spremljanje sprememb izdelka. Razvijalci so ocenili, da aktivnost izvajajo redko, kljub številnim priložnostim. Pogostost izvajanja smo želeli preveriti tudi z meritvami (M21). Rezultati so prikazani na sliki 4.6. Tudi meritve so pokazale, da se izvajanja



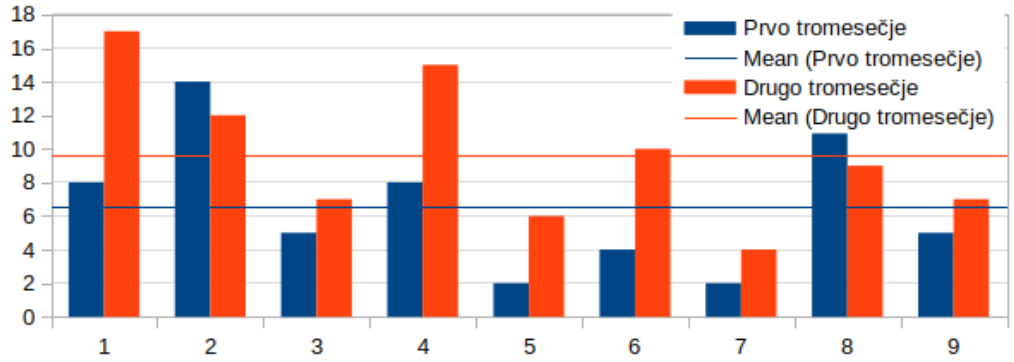
Slika 4.6: Pogostost označevanja produkcijskih različic za posameznega razvijalca (M21, podjetje A).

aktivnosti drži približno polovica razvijalcev. Razvijalci, ki so označevali produkcijske različice, so v povprečju to storili manj kot enkrat mesečno, priložnosti za to pa je po mnenju tehničnega osebja, več kot deset na mesec. Aktivnost je relativno nizko ocenjena tudi iz ekonomskega vidika. Kljub temu da razvijalci in vodstvo ne pripisujejo omenjeni aktivnosti večjega pomena, je tehnično osebje mnenja, da je treba izboljšati sprejetost te aktivnosti, saj

pripomore k lažjemu sledenju sprememb izdelka.

Izvajanje strokovnih pregledov (A15) se v podjetju uporablja za pregled izvorne kode s strani drugih programerjev. Namen te aktivnosti je deljenje znanja in preprečevanje napak. Razvijalci prek strokovnih pregledov predstavijo svoje predloge ali popravke, ki jih avtor kode sprejme ali zavrne z obrazložitvijo. Če razvijalec, ki izvaja pregled, nima pripomb, je koda potrjena in pripravljena na izdajo. Aktivnost je ocenjena kot slabše sprejeta in manj učinkovita.

Iz meritev (M2) prikazanih na sliki 4.7 ni bilo mogoče razbrati uporabnih informacij za to aktivnost. Število vej je namreč malo več kot devet na

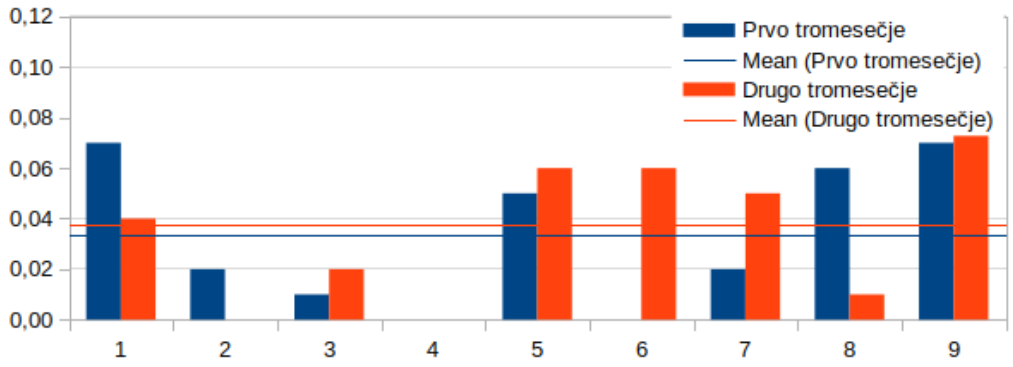


Slika 4.7: Število ustvarjenih vej v prvem in drugem tromesečju za posameznega razvijalca(merilo M2, podjetje A).

osebo, približno toliko pa je tudi priložnosti za izvedbo strokovnih pregledov. V primeru prenizkega števila vej bi lahko ocenili, da se aktivnost izvaja premalo, ko je pa to ustrezno veliko, pa ne moremo trditi nasprotno, saj ni nujno, da so z vsako novo vejo izvede tudi strokovni pregled. Razvijalci so ocenili, da aktivnost ne izboljša kakovosti dela, niti ni združljiva z njihovim delom. Razlog za to je časovna zahtevnost teh pregledov. Razvijalec, ki je zadolžen za recenzijo, mora pregledati in razumeti vsebino kode. To je lahko časovno zahtevna naloga, če je koda kompleksna. Ko smo tehničnemu osebju predstavili rezultate, so se odločili strokovne preglede prestaviti v aktivnost pregleda sprinta. S tem se omeji število teh pregledov, hkrati pa je prisoten

tudi avtor kode, ki lahko razloži njen pomen in omogoči lažje razumevanje.

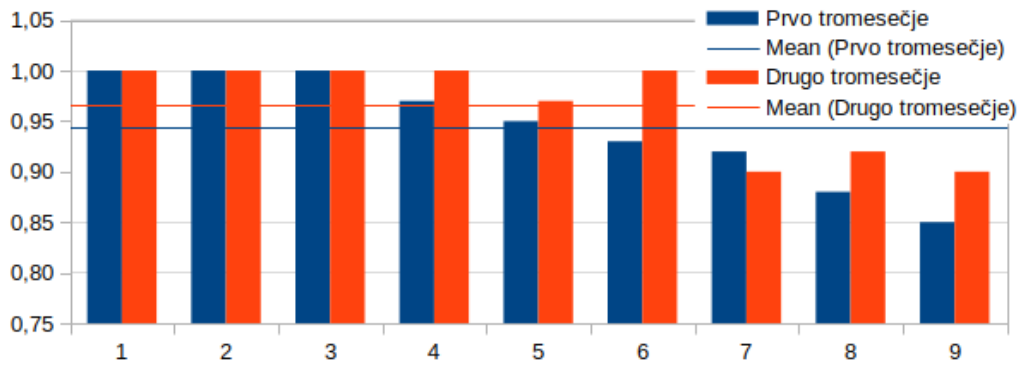
Testno vodeni razvoj (A11) so v podjetju vpeljali zato, da bi se razvijalci bolje držali zahtev pri razvoju novih funkcionalnostih. Aktivnost je v skladu z modernimi standardi in pristopi, vendar ni prilagodljiva razvijalcem. Vzrok za to je težavnost izvajanja te aktivnosti in je zato namenjena naprednejšim oziroma izkušenejšim razvijalcem. Tudi sociološki vidik je ocenjen z relativno slabo oceno. Če upoštevamo razpršenost opazimo, da so najnižje ocene tudi absolutno gledano slabe (pod nevtralno oceno 4). Razvijalci za trenutno stanje krivijo predvsem pomanjkanje dostopa do znanja na tem področju. Pogostost izvajanja smo preverili tudi z meritvami (M17) našega orodja in rudarjenjem procesov z orodjem Prom. Rezultati meritev so prikazani na sliki 4.8. Razvidno je, da so spremembe v testnih datotekah



Slika 4.8: Delež različic vsakega razvijalca, kjer so prisotne spremembe testov (merilo M17, podjetje A).

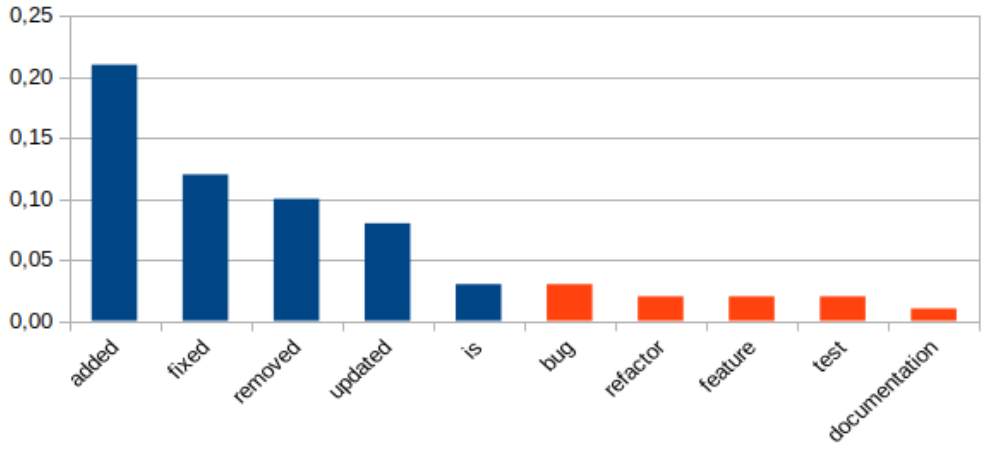
prisotne samo v treh odstotkih vseh različic. Z orodjem Prom smo analizirali dnevniške zapise na modulu *api*, tako kot pri aktivnosti dokumentiranja. Na sliki 4.5 je vizualizirana Petrijeva mreža, na podlagi katere, smo ugotovili, da se testiranje izvaja v 17 odstotkih primerov, kar je v primerjavi z drugimi moduli nadpovprečno, a po mnenju tehničnega osebja premalo. Pričakovali so, da se bo aktivnost izvajala v vsaj 40 odstotkih primerov. Po pregledu meritev je tehnično osebje ocenilo, da so razvijalci precenili pogostost izvajanja te aktivnosti.

Pisanje jedrnatih opisov verzij (A5) je namenjeno ohranjanju urejene zgodovine razvoja z namenom lažjega razumevanja preteklih sprememb, ki so vzrok za trenutno stanje kode. Aktivnost je slabše ocenjena iz sociološkega vidika. Glavni razlog za to je, da razvijalci ne vidijo koristi v izvajanju te aktivnosti. Tehnično osebje jo je ocenilo zelo pozitivno, z izjemo prilagodljivosti razvijalcem. Razvijalci ne vidijo kratkoročne koristi standardiziranih in jedrnatih opisov zaradi redkih situacij, kjer je treba pregledovati zgodovino. Po mnenju tehničnega osebja bo takih situacij vedno več, hkrati pa bi izboljšanje opisov omogočilo lažje ugotavljanje vsebin različic, s čimer bi olajšali prepoznavanje aktivnosti in posledično olajšali tudi analitiko vsebine različic. Na sliki 4.9 so prikazane meritve ustreznosti obsega opisov verzij (M3). V večini primerov (nad 97%) so se opisi izkazali kot primerno dolgi (vsaj en stavek, več kot 20 znakov). V podjetju želijo vsako različico označiti



Slika 4.9: Delež ustreznih obsegov opisov različic za vsakega razvijalca (merilo M3, podjetje A).

z ustrezno oznako, ki odraža izvedeno aktivnost (*bug*, *feature*, *refactor*, *documentation*, *test*, ipd.). Na sliki 4.10 so prikazane meritve ustreznosti uporabe oznak v opisih različic. Meritve predstavljajo deleže opisov različic v katerih se prikazana beseda pojavlja. Prvih pet meritev so najbolj pogoste besede v opisih, drugih pet pa najbolj pogoste pričakovane oznake. Označevanje aktivnosti v opisu ni prisotno, saj so najbolj pogoste besede glagoli in vezniki (prvih pet meritev na sliki 4.10), pričakovane oznake (drugih pet meritev na



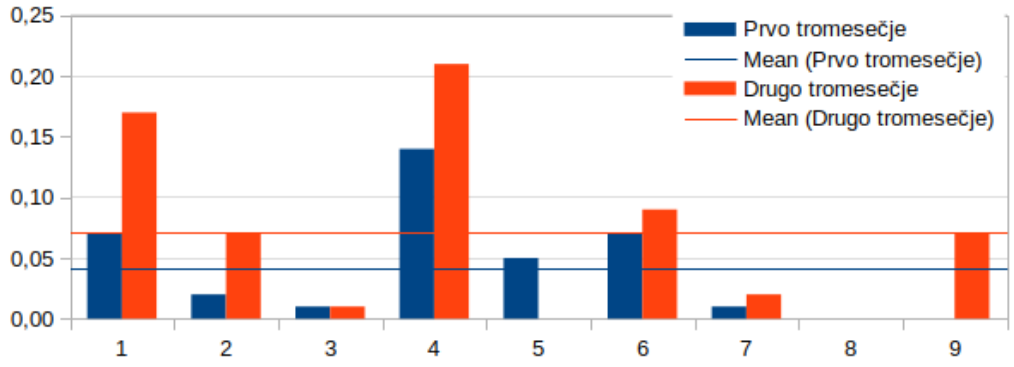
Slika 4.10: Najbolj pogoste besede in oznake v opisih verzij (merilo M4, podjetje A).

sliki 4.10) pa se pojavljajo le v nekaj odstotkov primerih. Tehnično osebje je rezultate meritev ocenilo kot uporabne, pogostost izvajanja so ocenili z nižjo oceno kot razvijalci. Pričakovali so namreč, da bo vsota deležev oznak slabih sto odstotkov, meritve pa kažejo, da jih razvijalci uporabljajo zgolj v 12%.

Ustvarjanje novih vej za novo funkcionalnost se v podjetju uporablja za ločevanje kode posameznih funkcionalnosti. S tem omogočijo vzporeden razvoj več funkcionalnosti in olajšajo prehod na razvoj drugih funkcionalnosti, če se prioritete spremenijo. Aktivnost je med razvijalci dobro sprejeta, ima pa slabšo oceno tehnične učinkovitosti. Kljub dobri sprejetosti je razvijalci ne uporabljajo vsakič, ko se za to pojavi priložnost. Pri pregledu meritev (M2), prikazanih na sliki 4.7, je tehnično osebje ugotovilo, da se pogostost izvajanja ujema z ocenami razvijalcev. Vsak razvijalec v povprečju ustvari vsaj tri nove veje mesečno, novih funkcionalnosti pa je po mnenju tehničnega osebja vsaj trikrat toliko. Pri tehničnem vidiku je ocena nižja pri karakteristikah ustreznosti za ekipo in za projekt. Po pregledu rezultatov sociološkega vidika je tehnično osebje ugotovilo, da je potrebna ustreznejša definicija obsega nove funkcionalnosti. Razvijalci so namreč mnenja, da prepogosto ustvarjanje različnih vej povzroča nepreglednost in nepotrebno dodatno delo. Nove funkcionalnosti so se zato odločili združiti v večje sklope,

ki so povezani glede na datum izdaje. Nova veja naj bi se ustvarjala za vsak tak sklop funkcionalnosti.

Pisanje testov (A6) je poleg razvoja novih funkcionalnosti in dokumentacije najbolje ocenjena aktivnost iz ekonomskega vidika. Zato je bilo treba raziskati rahlo podpovprečno sprejetost med razvijalci. Aktivnost obsega testiranje enot (angl. *unit testing*), pisanje integracijskih testov in funkcijskih testov. Pisanje testov je ključno za zagotavljanje kakovosti izdelka. Avtomatizacija testov prihrani čas, potreben za preverjanje delovanja obstoječih funkcionalnosti, katerih število se konstantno povečuje. S testi zagotovijo, da dodajanje novih funkcionalnosti ne pokvari delovanja obstoječih. Razvijalci so ocenili, da aktivnosti ne izvajajo vsakič, ko se za to pojavi priložnost, kljub temu pa je omenjena ocena relativno visoka. Pri pregledu meritev (A6) je tehnično osebje ugotovilo, da je aktivnost izvajana redkeje, kot so to ocenili razvijalci. Meritve so prikazane na sliki 4.11. Delež testov v vseh



Slika 4.11: Delež sprememb testnih datotek (merilo M8, Podjetje A).

spremembah datotek je približno sedem odstotkov. Kot potencialni razlog smo poudarili slabšo dostopnost znanja, ki je najslabše ocenjena karakteristika pri tej aktivnosti. Tudi prilagodljivost razvijalcem iz tehničnega vidika je ocenjena z relativno nizko oceno. Iz tega smo sklepali, da je pisanje testov naprednejša aktivnost, za katero je treba posedovati ustrezno znanje in izkušnje. Ker je aktivnost v podjetju stara manj kot eno leto, pričakujemo, da razvijalci potrebujejo čas, da se privadijo na redno pisanje testov. To

potrjuje tudi trend na sliki 4.7, kjer se je delež sprememb testov v trenutnem tromesečju potrojil v primerjavi s prejšnjim. Pri pregledu meritev je tehnično osebje ugotovilo, da se izvajanje aktivnosti med razvijalci zelo razlikuje. Kot razlog za to so podali razliko v znanju in izkušenosti razvijalcev na tem področju.

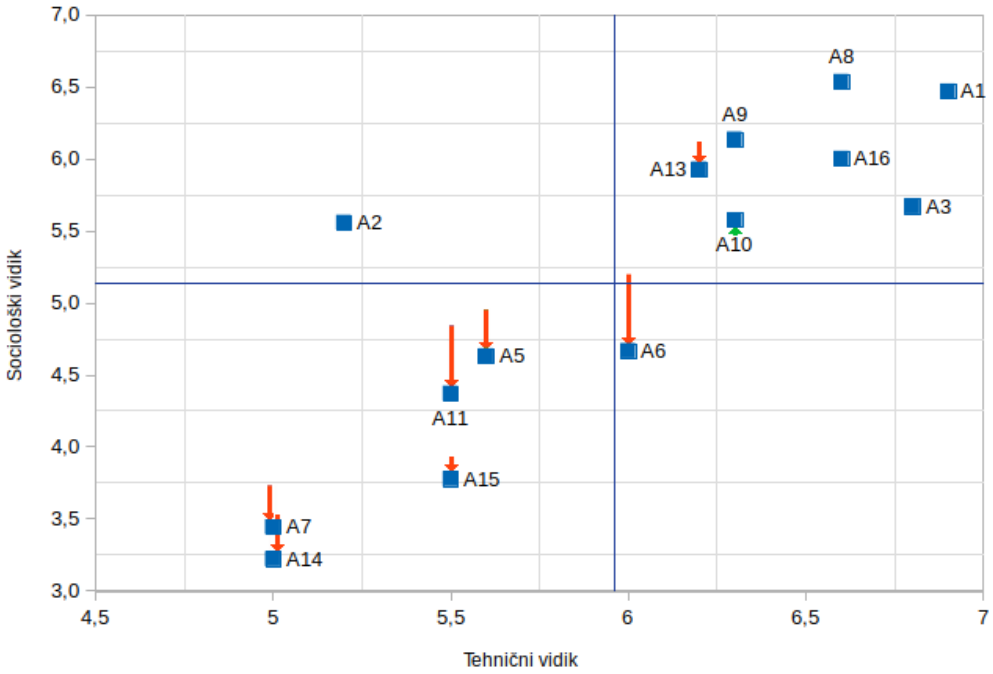
Po pregledu vseh aktivnosti, ki so ocenjene kot neustrezne, je tehnično osebje ocenilo tudi uporabnost meritev pri ostalih aktivnostih. Postopek ocenjevanja je opisan v poglavju 3.4. Rezultati ocenjevanja so prikazani v tabeli 4.3. Tretji stolpec predstavlja oceno uporabnosti meritev ne glede na

Aktivnost	Merilo	Ocena uporabnosti meritev	Ocena pogostost izvajanja (dnev.)	Spremenjena ocena soc. vidika	Ocena enotnosti pogostosti izvajanja
A1	M1	7	/	6,5 (0,0)	/
A2	M2	7	/	5,6 (0,0)	/
A3	/	/	/	5,7 (0,0)	/
A5	M3 in M4	6	3	4,6 (-0,3)	/
A6	M8	6	2	4,7 (-0,5)	2
A7	M6 in RP	5	2	3,4 (-0,3)	1
A8	M10	6	/	6,5 (0,0)	/
A9	M12	6	/	6,1 (0,0)	/
A10	M15	5	6	5,6 (0,1)	/
A11	M17 in RP	7	2	4,4 (-0,5)	2
A13	M19	4	5	5,9 (-0,2)	/
A14	M21	7	2	3,2 (-0,2)	1
A15	M2	4	3	3,8 (-0,2)	1
A16	/	/	/	6,0 (0,0)	/

Tabela 4.3: Ocene uporabnosti posameznih meril in spremenjene ocene sociološkega vidika (podjetje A).

to, ali so bile uporabljene pri ocenjevanju ali ne, četrti stolpec predstavlja podano oceno pogostosti izvajanja na podlagi dnevniških zapisov, peti stolpec predstavlja novo oceno sociološkega vidika, ki upošteva tudi oceno iz četrtega stolpca. Zadnji stolpec predstavlja oceno enotnosti pri izvajanju aktivnosti na podlagi dnevniških zapisov. Tehnično napredno osebje je sedem meril ocenilo kot uporabne. Od teh sedmih je šest meritev uporabilo za podajanje ocene izvajanja pogostosti aktivnosti. Na sliki 4.12 so prika-

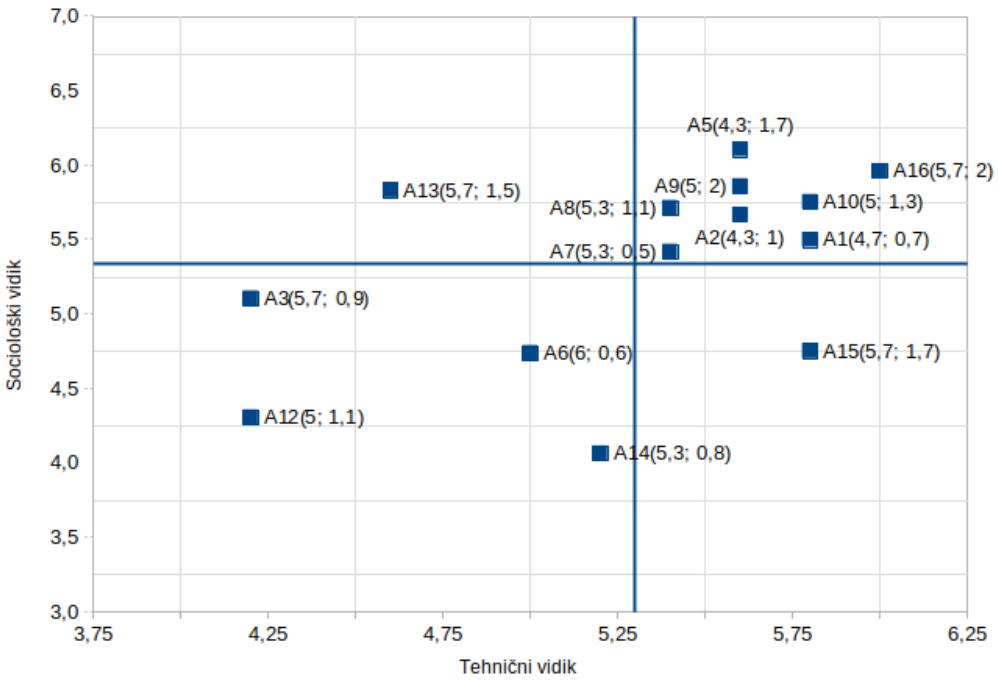
zane razlike med ocenami pred upoštevanjem analize dnevniških zapisov in po tem. Vektor rdeče barve pomeni, da se je ocena sprejetosti elementa po upoštevanju dnevniških zapisov poslabšala, vektor zelene barve pa pomeni da se je izboljšala.



Slika 4.12: Spremembe v ocenah po upoštevanju rezultatov analize dnevniških zapisov (podjetje A).

4.2.2 Podjetje B

Zaposleni v podjetju B so ocenjevali 13 aktivnosti, ki so predstavljene v tabeli 4.1.4. Sociološki vidik je ocenjevalo šest razvijalcev, tehnični je ocenjeval eden, ekonomski pa samo direktor podjetja. Rezultati evalvacije so prikazani na sliki 4.13. Povprečne ocene karakteristik vseh treh vidikov so prikazane v tabeli 4.4.



Slika 4.13: Ocene sociološkega, tehničnega in ekonomskega vidika v podjetju B.

Rezultati so razdeljeni glede na relativno oceno. Kot neustrezne iz obeh vidikov so bile ocenjene aktivnosti A3, A6, A12 in A14. Aktivnost A13 je bila ocenjena kot bolje sprejeta a manj učinkovita od povprečja. Nasprotna situacija je z aktivnostjo A15, ki je bila ocenjena kot tehnično ustrezna in slabše sprejeta. Iz rezultatov je razvidna tudi večja razpršenost ocen pri elementih A16 in A9. Tehnično osebje je bilo treba opozoriti na taka odstopanja. V nadaljevanju smo obravnavali neustrezne aktivnosti. Začeli smo s tistimi, ki

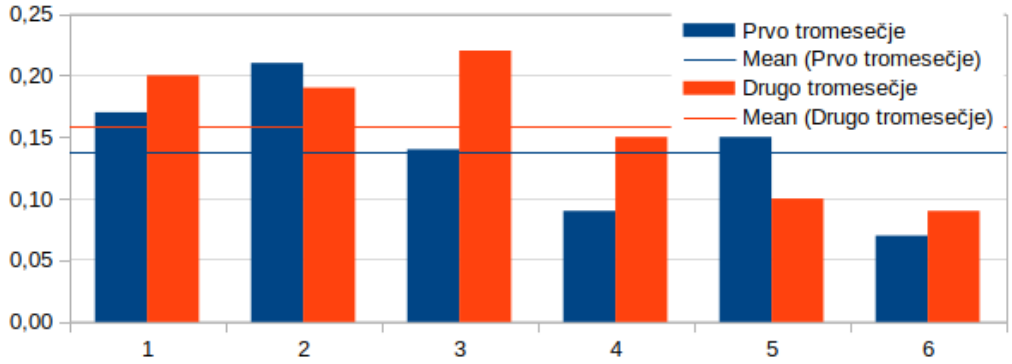
Sociološki vidik	Pogostost uporabe	Izboljša kakovost dela	Združljivo z delom	Prostovoljnost	Dostopnost znanja	Skupna ocena
A1	6,5	5,7	5,0	4,3	6,0	5,5
A2	5,5	5,4	6,2	5,7	5,5	5,7
A3	5,3	5,4	4,5	4,9	5,4	5,1
A5	6,3	5,8	6,3	6,1	6,0	6,1
A6	4,0	6,3	4,7	5,5	3,2	4,7
A7	6,3	6,3	5,5	5,7	3,3	5,4
A8	6,7	5,5	6,2	5,7	4,5	5,7
A9	5,3	6,5	6,3	5,5	5,7	5,9
A10	5,3	6,3	5,7	6,8	4,7	5,8
A12	2,7	5,5	5,7	3,7	4,0	4,3
A13	5,5	6,7	6,3	5,3	5,3	5,8
A14	4,7	2,7	3,3	3,3	6,3	4,1
A15	6,0	6,2	4,2	3,7	3,7	4,8
A16	6,3	5,7	5,5	5,7	6,7	6,0
Tehnični vidik	Pogostost priložnosti	Ustreznost za ekipo	Ustreznost za projekt	Moderni standardi	Prilagodljivost razvijalcem	Skupna ocena
A1	6	6	6	7	4	5,8
A2	6	5	5	6	6	5,6
A3	5	5	4	3	4	4,2
A5	5	6	6	7	4	5,6
A6	5	6	6	6	2	5
A7	4	7	7	6	3	5,4
A8	6	7	6	5	3	5,4
A9	5	5	7	6	5	5,6
A10	6	5	6	6	6	5,8
A12	5	3	5	4	4	4,2
A13	7	5	5	4	2	4,6
A14	5	7	6	6	2	5,2
A15	5	6	6	5	7	5,8
A16	6	6	6	5	7	6
Vodstveni vidik	Vpliv na produkt	Stroški	Strateški cilij			Skupna ocena
A1	4	5	5			5,0
A2	4	4	5			5,0
A3	7	4	6			6,0
A5	4	4	5			5,0
A6	7	6	5			5,0
A7	6	5	5			5,0
A8	7	3	6			6,0
A9	5	5	5			5,0
A10	6	5	4			4,0
A12	7	4	4			4,0
A13	7	5	5			5,0
A14	5	7	4			4,0
A15	6	7	4			4,0
A16	7	5	5			5,0

Tabela 4.4: Povprečne ocene posameznih vidikov in karakteristik v podjetju B.

imajo najnižjo oceno. Pri interpretaciji rezultatov nam je pomagalo tehnično osebje.

Testiranje vseh funkcionalnosti (A12) je podpovprečno ocenjena aktivnost tako iz sociološkega kot tudi iz tehničnega vidika. Aktivnost se v pod-

jetju uporablja za zagotavljanje pokritosti kode s testi. Za razliko od testno vodenega pristopa pri tej aktivnosti ni pomembno zaporedje pisanja testov in razvoja funkcionalnosti. Razvijalci so pogostost izvajanja, ko se pojavi priložnost, ocenili z zelo nizko oceno (2,7). Kot glavni vzrok so podali slabšo dostopnost znanja. Pri pregledu meritev (M17), ki so prikazane na sliki 4.14, je tehnično osebje ugotovilo, da se aktivnost izvaja bolj pogosto, kot so to ocenili razvijalci. Slabše ocene pogostost izvajanja s strani razvijalcev so

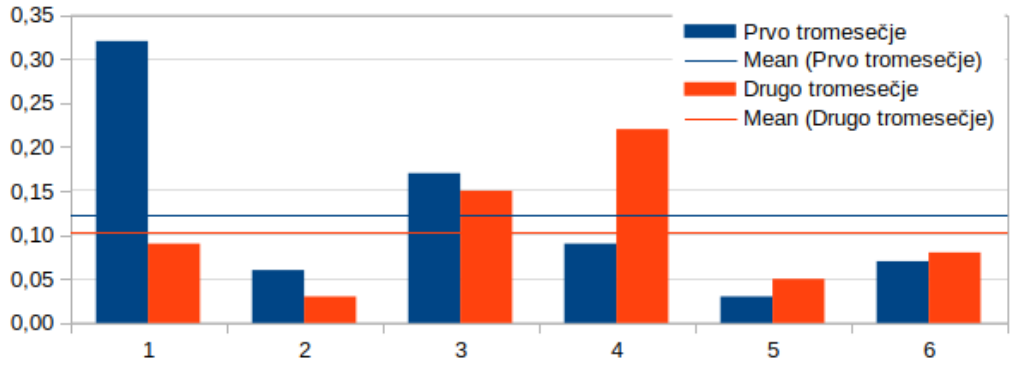


Slika 4.14: Delež različic s spremembami testnih datotek (M17, podjetje B).

po mnenju tehničnega osebja posledica šibke definiranosti zahtevanega obsega testov. Razvijalci so precenili zahteve vodstva pri pogostosti izvajanja. Tehnično osebje je ugotovilo, da je za boljšo sprejetost te aktivnost treba bolje definirati zahtevani obseg testiranja.

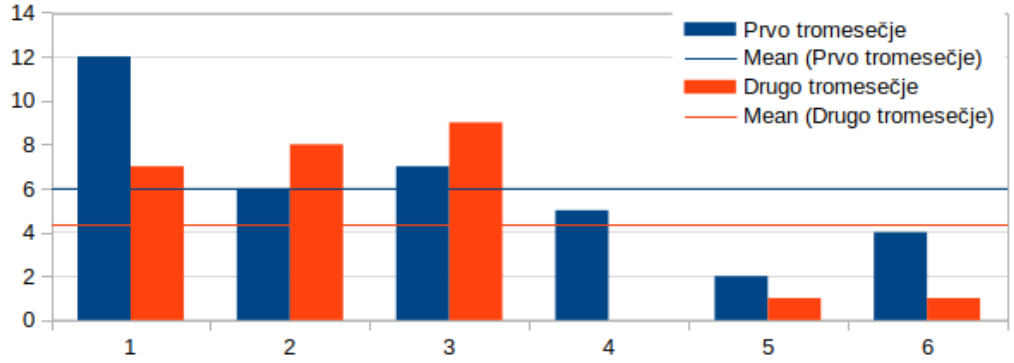
Pisanje testov (A6) ni daleč od povprečja po sprejetosti in tehnične učinkovitosti, a ima zaradi visoke ocene iz ekonomskega vidika prednost pri obravnavi. Razvijalci so najslabšo oceno dali karakteristikam pogostosti uporabe in dostopnosti znanja. Pri pregledu meritev (M8), ki so prikazane na sliki 4.15, je tehnično osebje ugotovilo, da se aktivnost izvaja dovolj pogosto za trenuten projekt.

Razvijalci so tako kot pri aktivnosti A12 precenili pogostost priložnosti za izvajanje te aktivnosti. Razlog za slabšo oceno sprejetosti te aktivnosti je podoben kot pri aktivnosti A12.



Slika 4.15: Delež sprememb testnih datotek (M8, podjetje B).

Označevanje produkcijskih različic (A14) se v podjetju uporablja na enak način, kot v podjetju A. Razvijalci so ocenili, da imajo na voljo vse potrebno znanje za izvajanje te aktivnosti in jo delno tudi izvajajo. Menijo pa, da aktivnost ne izboljša njihove kakovosti dela in da ni združljiva z njihovim delom. Tehnično osebje je drugačnega mnenja, saj so zelo visoko ocenili ustreznost za ekipo in za projekt. Po pregledu meritev (M21) prikazanih na sliki 4.16 je tehnično osebje ugotovilo, da razvijalci niso enotni pri pogostosti izvajanja. Polovica razvijalcev ustrezno pogosto izvaja aktivnost, medtem



Slika 4.16: Pogostost označevanja produkcijskih različic (M21, podjetje B).

ko jo druga polovica izvaja preredko. Kot morebiten razlog za tako stanje so navedli pogostost manjših popravkov, ki jih razvijalci implementirajo v izdelek. Označevanje vsakega majhnega popravka se razvijalcem zdi časovno

potratno, zato s časom opustijo izvajanje te aktivnosti. Razvijalci, ki ne implementirajo pogostih manjših popravkov so aktivnost bolje sprejeli in jo tudi izvajajo vsakič, ko se pojavi priložnost.

Načrtovanje sprinta (A3) je med razvijalci dobro sprejeta aktivnost, a ima slabšo oceno iz tehničnega vidika. Kot glavni razlog je tehnično osebje podalo neskladnost z modernimi standardi. Tehnično osebje je pojasnilo, da za načrtovanje ne uporabljajo specializiranega orodja, kar je tudi razlog za tako nizko oceno. Za to aktivnost ni bilo na voljo dnevniških zapisov.

Izvajanje strokovnih pregledov (A15) je ocenjena kot tehnično ustrezna aktivnost, med razvijalci pa je slabše sprejeta. Kot glavni vzrok za slabšo sprejetost so navedli dostopnost znanja za izvajanje te aktivnosti. Menijo tudi, da pregledi v trenutni obliki niso združljivi z njihovim delom. Tehnično osebje je ugotovilo, da se pregledi kode zelo pogosto uporabljajo. Razvijalec, ki pregleduje, mora pogosto preusmerjati pozornost iz svojega trenutnega dela na kodo, ki je predmet pregleda. S pogostimi prekinitvami dela se zmanjša produktivnost razvijalcev. Tehnično osebje je merilo M2 ocenilo kot neustrezno, ker ne odraža dejanske pogostosti izvajanj strokovnih pregledov.

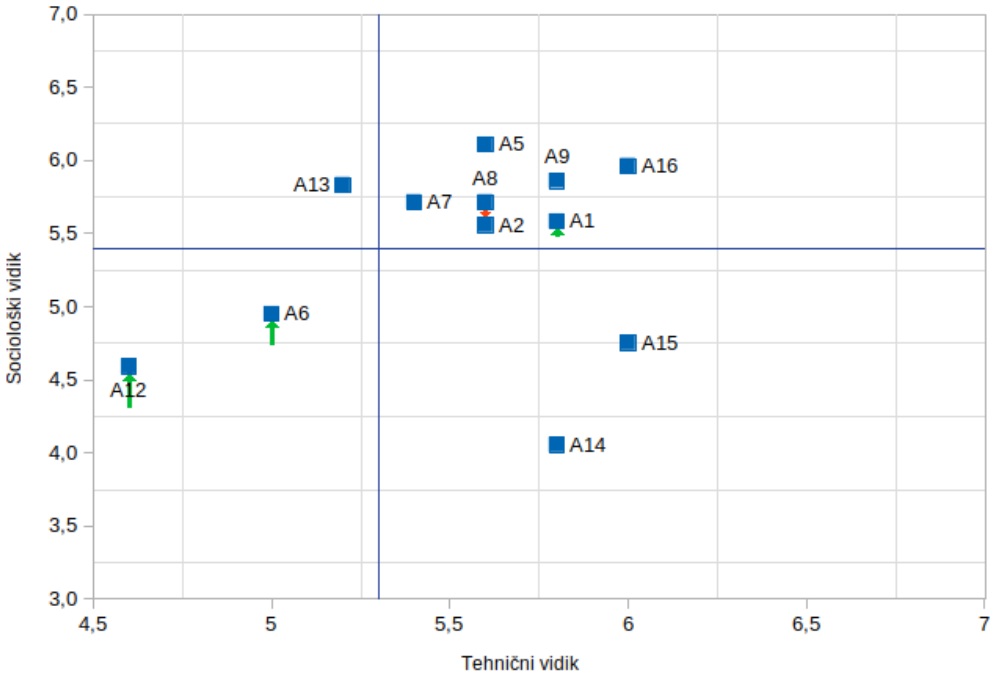
Testiranje pred shranjevanjem kode (A13) je med razvijalci dobro sprejeto a ima slabšo oceno iz tehničnega vidika. Kot glavno pomanjkljivost tehnično osebje vidi slabo prilagodljivost razvijalcem. Pred shranjevanjem kode so namreč definirani postopki, ki se jih mora razvijalec držati ne glede na svoje znanje in izkušnje. Merilo M19 so ocenili kot neprimerno za uporabo v njihovem primeru, saj je zaporedno shranjevanje manjših zaporednih različic ustaljena praksa in ni kazalnik pomanjkanja testiranja.

Po pregledu vseh aktivnosti, ki so ocenjene kot neustrezne, je tehnično osebje ocenilo tudi uporabnost meritev pri ostalih aktivnostih. Postopek ocenjevanja je opisan v poglavju 3.4. Rezultati ocenjevanja so v tabeli 4.5. Tehnično osebje je za osem aktivnosti podalo oceno pogostosti izvajanja aktivnosti na podlagi analize dnevniških zapisov. To pomeni, da so pri teh aktivnostih prepoznali rezultate meritev kot bolj primeren kazalnik pogostosti uporabe kot ocene razvijalcev. Na sliki 4.17 so prikazane razlike med

Aktivnost	Merilo	Ocena uporabnosti meritev	Ocena pogostost izvajanja (dnev.)	Spremenjena ocena soc. vidika	Ocena enotnosti pogostosti izvajanja
A1	M1	7	6	5,6 (0,1)	/
A2	M2	6	5	5,6 (-0,1)	/
A3	/	/	/	5,1 (0,0)	/
A5	M3	5	/	6,1 (0,0)	/
A6	M8	7	6	5,0 (0,2)	/
A7	/	/	/	5,4 (0,0)	/
A8	M10	5	/	5,7 (0,0)	/
A9	M12	5	/	5,9 (0,0)	/
A10	M15	4	6	5,8 (0,0)	/
A12	M17	7	6	4,6 (0,3)	/
A13	M19	2	/	5,8 (0,0)	/
A14	M21	6	4	4,1 (0,0)	1
A15	M2	3	/	4,8 (0,0)	/
A16	/	/	/	6,0 (0,0)	/

Tabela 4.5: Ocene uporabnosti posameznih meril in spremenjene ocene sociološkega vidika (podjetje B).

ocenami pred upoštevanjem analize dnevniških zapisov in po tem.



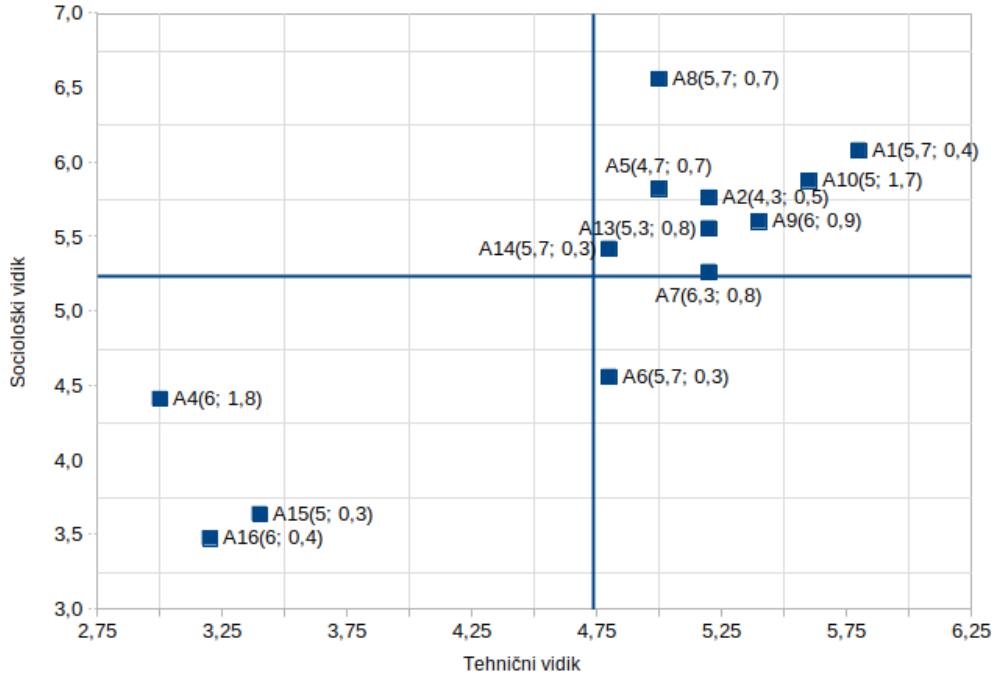
Slika 4.17: Spremembe v ocenah po upoštevanju rezultatov analize dnevniških zapisov (podjetje B).

4.2.3 Podjetje C

Zaposleni v podjetju C so ocenjevali 12 aktivnosti, ki so predstavljene v tabeli 4.1.4. Sociološki vidik je ocenjevalo šest razvijalcev, tehnični je ocenjeval eden, ekonomski pa vodja oddelka. Rezultati evalvacije so prikazani na sliki 4.18. Povprečne ocene karakteristik vseh treh vidikov so prikazane v tabeli 4.6.

Rezultati so razdeljeni v štiri kvadrante glede na relativno oceno. Kot neustrezne iz obeh vidikov so bile ocenjene aktivnosti A4, A15 in A16. Aktivnost A6 je bila ocenjena kot tehnično učinkovita, a slabše sprejeta od povprečja. V nadaljevanju smo obravnavali vse štiri neustrezne aktivnosti

Poročanje o napredku (A16) je najslabše sprejeta aktivnost kljub visoki oceni iz ekonomskega vidika. Tudi tehnična učinkovitost te aktivnosti je slabša od povprečja. Kot glavni razlog za slabo sprejetost so razvijalci oce-



Slika 4.18: Ocene sociološkega, tehničnega in ekonomskega vidika v podjetju C.

nili slabo združljivost z delom in dostopnost znanja. Tehnično osebje je kot razloge za slabo učinkovitost navedlo neskladnost s sodobnimi standardi in pristopi. Razlog za slabo sprejetost in neučinkovitost aktivnosti je odvisnost njenega izvajanja od zunanjih naročnikov. Razvijalci se morajo prilagajati obliki poročanja, ki jo zahteva naročnik. Te se pa pogosto razlikujejo, zaradi česar se razlikujejo tudi orodja, ki se uporabljajo za poročanje. Dnevniških zapisov za omenjena orodja nismo pridobili, zato smo analizo meritev za to aktivnost preskočili.

Izvajanje strokovnih pregledov (A15) ima podobno oceno sociološkega in tehničnega vidika kot aktivnost A16. Razvijalci so kot glavni razlog navedli slabo združljivost z delom. Tehnično osebje je ocenilo, da je malo priložnosti za izvajanje te aktivnosti, kot glavni razlog za to so izpostavili neskladnost z modernimi standardi in neustreznost za ekipo. Aktivnost ni ustrezna za ekipo zaradi raznolikosti platform, na katerih razvijalci delajo. Razvijalci so

Sociološki vidik	Pogostost uporabe	Izboljša kakovost dela	Združljivo z delom	Prostovoljnost	Dostopnost znanja	Skupna ocena
A1	4,3	6,6	6,4	6,4	6,7	6,1
A2	4,7	5,8	5,3	6,3	6,7	5,8
A4	5,3	4,3	3,3	4,5	4,7	4,4
A5	5,7	5,7	6	5,7	6,1	5,8
A6	2,1	5,7	4,7	5,7	4,7	4,6
A7	5,3	4,7	5,7	5,3	5,3	5,3
A8	6,7	6,7	6,8	6,6	6,1	6,6
A9	6,1	5,7	6,1	5,3	4,8	5,6
A10	6	5,7	6,1	6,1	5,5	5,9
A13	5,5	5,3	5,7	5,8	5,5	5,6
A14	5,3	5,1	4,7	5,7	6,3	5,4
A15	2,7	5,7	2,8	3,3	3,7	3,6
A16	5,1	4,3	2,3	3,3	2,3	3,5
Tehnični vidik	Pogostost priložnosti	Ustreznost za ekipo	Ustreznost za projekt	Moderni standardi	Prilagodljivost razvijalcem	Skupna ocena
A1	7	7	7	6	2	5,8
A2	4	5	4	6	7	5,2
A4	6	1	2	2	4	3
A5	6	6	4	7	2	5
A6	5	4	5	6	4	4,8
A7	6	5	5	5	5	5,2
A8	7	5	7	3	3	5
A9	6	5	6	6	4	5,4
A10	4	7	5	5	7	5,6
A13	5	6	4	6	5	5,2
A14	4	6	6	6	2	4,8
A15	3	3	3	3	5	3,4
A16	5	2	3	2	4	3,2
Vodstveni vidik	Vpliv na produkt	Stroški	Strateški cilij			Skupna ocena
A1	4	4	6			4,7
A2	4	4	5			4,3
A4	6	6	6			6
A5	4	4	6			4,7
A6	5	6	6			5,7
A7	6	6	7			6,3
A8	7	4	6			5,7
A9	7	4	7			6
A10	5	3	7			5
A13	5	4	7			5,3
A14	6	5	6			5,7
A15	5	4	6			5
A16	6	6	6			6

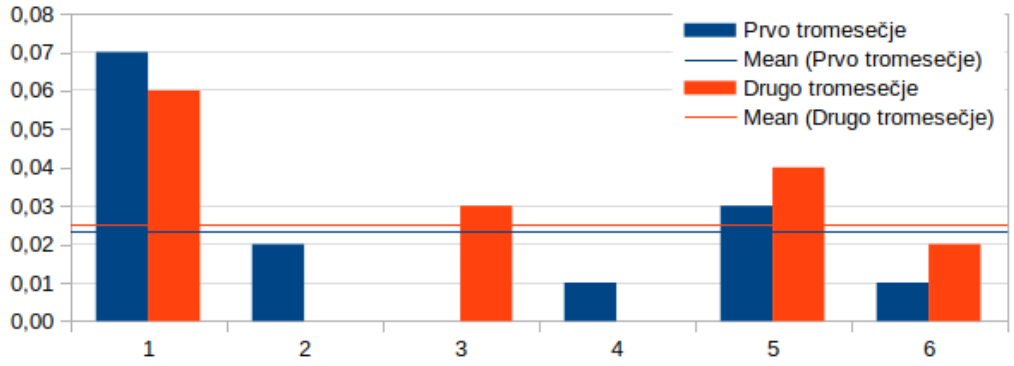
Tabela 4.6: Povprečne ocene posameznih vidikov in karakteristik v podjetju C.

specializirani za svoja področja in zaradi tega ne vidijo dodane vrednosti pri pregledovanju kode sorazvijalcev. Tehnično osebje je ocenilo, da iz meritev M2 ne morejo soditi o pogostosti izvajanja aktivnosti.

Zajem zahtev (A4) je ocenjena kot manj učinkovita in slabše sprejeta aktivnost. Razvijalci so kot glavni razlog za slabo sprejetost navedli slabšo

združljivost z delom. Tehnično osebje je kot razloge za slabšo učinkovitost navedlo pomanjkanje skladnosti z modernimi standardi in neustreznost za ekipo. Vzorec razlogov je podoben kot pri aktivnosti A16. Pridobljene ocene so smiselne, saj sta obe aktivnosti odvisni od naročnikov. Ugotovili smo, da je glavna težava prilagajanje oblikam načrtovanja izdelka in poročanja, kot ju določa naročnik.

Pisanje testov (A6) je v primerjavi z drugima dvema podjetjema ocenjena z nekoliko slabšo oceno iz ekonomskega vidika. Ocenjena je bila kot slabše sprejeta, a tehnično ustrezna aktivnost. Pogostost uporabe je ocenjena z nizko oceno, kot glavni razlog so razvijalci navedli slabo združljivost z delom. Vzrok za to je, da večinoma izdelujejo prototipe, kjer je hitrost razvoja na prvem mestu. Pisanje testov v nekaterih primerih predstavlja oviro, saj se je treba prilagajati zahtevam naročnikom, ki se pogosto spreminjajo. To je tudi razlog za nekoliko nižjo oceno iz ekonomskega vidika. Pogostost izvajanja smo preverili tudi z meritvami (M8). Rezultati meritev so prikazani na sliki 4.19. Tehnično osebje je ocenilo, da se aktivnost izvaja



Slika 4.19: Delež sprememb testnih datotek (Podjetje C).

pogosteje, kot so to ocenili razvijalci. Razlog za odstopanje v ocenah je podoben, kot pri podjetju B. Razvijalci nimajo jasno definiranega zahtevanega obsega testiranja.

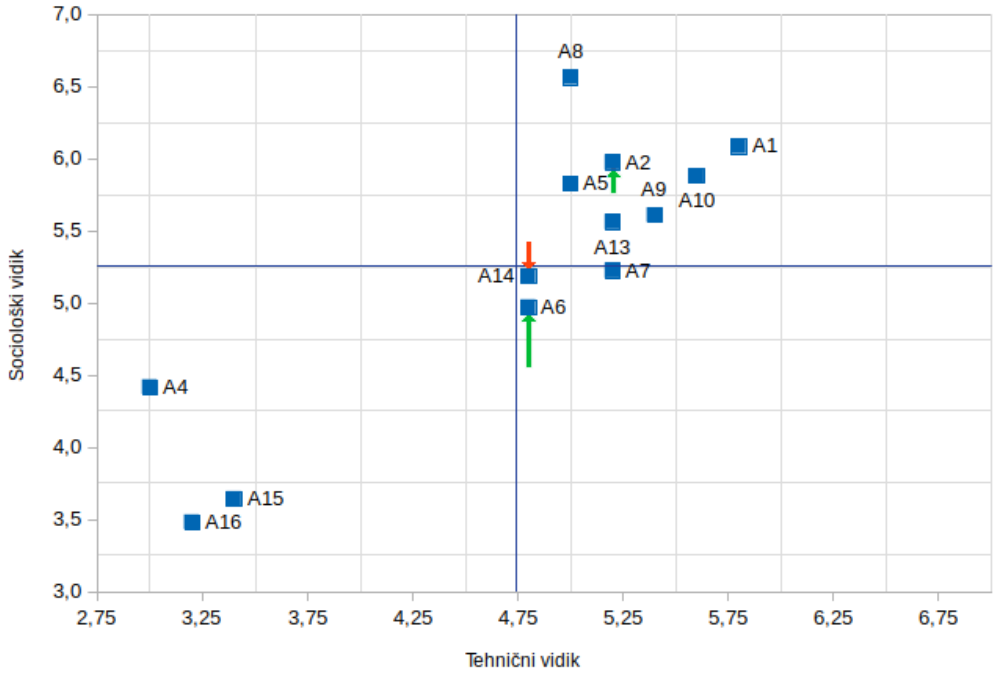
Po pregledu vseh neustreznih aktivnosti je tehnično osebje podalo oceno uporabnosti meritev in na podlagi teh meritev tudi oceno za relativno po-

gostost izvajanja aktivnosti. Postopek ocenjevanja je opisan v poglavju 3.4. Rezultati ocenjevanja meritev in njihovega vpliva so prikazani v tabeli 4.7. Spremembe v ocenah sociološkega vidika so vizualizirane s posodobljenim razsevnim diagramom na sliki 4.20. Tehnično osebje je osem meril ocenilo

Aktivnost	Merilo	Ocena uporabnosti meritev	Ocena pogostost izvajanja (dnev.)	Spremenjena ocena soc. vidika	Ocena enotnosti pogostosti izvajanja
A1	M1	7	/	6,1 (0,0)	/
A2	M2	6	7	6,0 (0,2)	/
A4	/	/	/	4,4 (0,0)	/
A5	M3	5	/	5,8 (0,0)	/
A6	M8	7	7	5,0 (0,4)	/
A7	M6	6	5	5,2 (0,0)	/
A8	M10	6	/	6,6 (0,0)	/
A9	M12	5	/	5,6 (0,0)	/
A10	M15	5	/	5,9 (0,0)	/
A13	M19	4	/	5,6 (0,0)	/
A14	M21	7	4	5,2 (-0,2)	/
A15	M2	4	/	3,6 (0,0)	/
A16	/	/	/	3,5 (0,0)	/

Tabela 4.7: Ocene uporabnosti posameznih meril in spremenjene ocene sociološkega vidika (podjetje C).

kot uporabne. V novih rezultatih lahko opazimo, da se je zaradi zmanjšanja pogostosti izvajanja spremenila ocena aktivnosti A14 in A7 na podpovprečno.



Slika 4.20: Spremembe v ocenah po upoštevanju rezultatov analize dnevniških zapisov (podjetje C).

4.3 Skupni rezultati

V vseh treh podjetjih se je tehnično osebje odločilo upoštevati meritve pridobljene z analizo dnevniških zapisov orodij za nadzor izvorne kode. Na podlagi meritev so podali oceni pogostosti izvajanja aktivnosti in enotnosti razvijalcev pri ocenjevanju. V tabeli 4.8 so prikazane ocene uporabnosti dnevniških zapisov pri ocenjevanju relativne pogostosti izvajanja aktivnosti. Ocene so pridobljene na podlagi pristopa, opisanega v poglavju 3.4. Po Likertovi lestvici, prikazani na sliki 3.3, so pozitivne vse ocene, katerih vrednost je večja od 4. Od trinajstih aktivnosti, ki smo jih preverjali z dnevniškimi zapisi, so se meritve izkazale kot uporabne pri devetih. Merjenje aktivnosti A14 in A10 se je izkazalo kot delno uporabno, saj je pri obeh kljub eni nevtralni oceni prisotna vsaj ena pozitivna. Meritve aktivnosti A13 in A14 smo prepoznali kot delno neuporabne, ker imata po dve nevtralni oceni in eno negativno.

Id	Aktivnost	Podj. A	Podj. B	Podj. C	Povpr. ocena
A1	Shranjevanje različic z orodjem Git	7	7	7	7
A2	Ustvarjanje nove veje za novo funkcionalnost	7	6	6	6,33
A5	Pisanje jedrnatih opisov različic	6	5	5	5,33
A6	Pisanje testov	6	7	7	6,67
A7	Pisanje dokumentacije	5	6	6	5,66
A8	Pisanje novih funkcionalnosti	6	5	6	5,67
A9	Konfiguriranje projekta	6	5	5	5,33
A10	Preurejanje kode	5	4	5	4,67
A11	Upoštevanje testno vodenega razvoja	7	/	/	7
A12	Testiranje vseh funkcionalnosti	/	7	/	7
A13	Testiranje pred shranjevanjem kode	4	2	4	3,3
A14	Uporaba oznak za produkcijske različice	7	6	4	5,67
A15	Izvajanje strokovnih recenzij	4	3	4	3,67

Tabela 4.8: V tabeli so prikazane ocene uporabnosti dnevniških zapisov v vseh treh podjetjih.

Tehnično osebje podjetij je meritvi označila kot dvoumni, saj lahko odražata izvajanje več različnih aktivnosti. V obeh primerih bi potrebovali bolj kompleksna merila, a jih zaradi omejitve dostopa do dnevniških zapisov nismo uporabili.

Tehnično osebje je bilo med analizo rezultatov dnevniških zapisov seznanjeno z ocenami sociološkega, tehničnega in ekonomskega vidika. Pri analizi pozitivno ocenjenih meritev aktivnosti je tehnično osebje imelo možnost podati tudi svojo oceno pogostosti izvajanja aktivnosti s katero so lahko vplivali na oceno razvijalcev. Tehnično osebje vseh treh podjetjih je na podlagi rezultatov analize dnevniških zapisov skupaj podalo novo oceno pogostosti izvajanj osemnajstkrat (tabele 4.3, 4.5 in 4.7). Po rezultatih sklepamo, da je tehnično osebje prepoznalo podane meritve kot bolj verodostojen dokaz o pogostosti izvajanja od ocen razvijalcev. Kot najbolj uporaben primer se je izkazala ocena pogostosti pisanja testov. V vseh treh podjetjih se je ocena razvijalcev razlikovala od tiste, ki jo je podalo tehnično osebje na podlagi meritev. Glavni vzrok za odstopanja meritev od ocen razvijalcev je po mnenju tehničnega osebja različno zaznavanje števila priložnosti za izvajanje pri razvijalcih. Če razvijalec nima pravilne predstave o številu priložnosti za izvajanje določene aktivnosti, obstaja verjetnost, da bo posledično neustrezno ocenil tudi pogostost izvajanja aktivnosti glede na število priložnosti. Motivacije za zavestno ponarejanje ocen ni bilo zaradi anonimnosti rezultatov in jasne obrazložitve razvijalcem, da je predmet ocenjevanja ustreznost procesov in ne produktivnost zaposlenih.

Ocene razvojnih procesov, obogatene z interpretacijami rezultatov s strani tehničnega osebja, smo predali vodstvom podjetij. Zaradi obsega in podrobnosti poročil smo rezultate v tem poglavju strnili in izpostavili najbolj kritične. Opisali smo jih v obliki predlogov, ki so našteti v nadaljevanju.

1. Predlog - bolj natančna opredelitev zahtevanega obsega izvajanja za aktivnosti pri katerih smo ugotovili večja odstopanja med ocenami razvijalcev in ocenami pridobljenimi z upoštevanjem meritev. Omenjeni predlog velja za aktivnosti testiranja v vseh treh podjetjih in pisanje

dokumentacije v podjetju A.

2. Predlog - povečanje prenosa znanja znotraj podjetja. Visoka razpršenost ocen sprejetosti je lahko kazalnik, da del razvijalcev bolje razume in upošteva podane smernice pri izvajanju aktivnosti od drugih. Primeri takih aktivnosti so pisanje testov, testno vodeni pristop in pisanje dokumentacije v podjetju A, zajem zahtev in izvajanje strokovnih pregledov v podjetju C ter označevanje produkcijskih različic v podjetjih A in B. Kot eno izmed možnosti za izboljšanje prenosa znanja smo predlagali razširitev že obstoječih strokovnih pregledov tako, da vključujejo tudi pregled izvajanja naštetih aktivnosti.

3. Predlog - prilagajanje zahtevanega obsega izvajanja določenih aktivnosti razvijalcem. Prevelik obseg ekonomsko manj učinkovitih aktivnosti lahko razvijalce ovira pri izvajanju ekonomsko bolj učinkovitih in s tem zmanjša njihovo produktivnost. Primer prepogosto zahtevanega izvajanja je označevanje verzij in ustvarjanje vej za nove funkcionalnosti v podjetjih A in strokovni pregledi v podjetju B.

4. Predlog - v primeru slabe ocene dostopnosti znanja, smo vodstvom predlagali organiziranje delavnic in namenitev dela delovnega časa razvijalcev raziskovanju ter učenju tehnik in dobrih praks za izbrane aktivnosti. Primeri takih aktivnosti so testno vodeni pristop v podjetju A, pisanje dokumentacije v podjetjih A in B, izvajanje strokovnih pregledov v podjetju B in poročanje o napredku v podjetju C.

Vodstva vseh treh podjetij so rezultate pregledala in jih ocenila kot dobro podlago za sprejemanje nadaljnjih odločitev o izboljšavah razvojnih procesov. Vodstva so tudi mnenja, da je uporaba dnevniških zapisov pri ocenjevanju povečala njihovo zaupanje v rezultate evalvacije. V podjetjih A in B je upoštevanje meritev dnevniških zapisov povečalo natančnost ocenjevanja sprejetosti, vendar ne v taki meri, da bi vplivala na prioriteto obravnave aktivnosti. V podjetju C pa je upoštevanje meritev privedlo do novega spoznanja. Po upoštevanju ocen pridobljenih z analizo dnevniških zapisov sta

aktivnosti pisanja dokumentacij in označevanja produkcijskih različic padli v skupino slabše sprejetih (slika 4.20). S tem se je spremenila tudi prioriteta obravnave neustreznih aktivnosti. Pisanje dokumentacije, ki je bila prej ocenjena kot bolje sprejeta aktivnost je po spremembi ocene padla v skupino manj sprejetih. Čeprav je aktivnost pisanja dokumentacije še zmeraj ocenjena kot bolje sprejeta od aktivnosti pisanja testov, je zaradi višje ocene iz ekonomskega vidika bolj kritična za izboljšavo.

Treba se je zavedati, da razvijalci podajo ocene sprejetosti aktivnosti na podlagi lastnega zaznavanja. Zaznana pogostost izvajanja aktivnosti pa se lahko razlikuje od dejanske pogostosti izvajanja. Razvijalci so namreč lahko prepričani, da so aktivnost dobro sprejeli, vendar je še vedno ne izvajajo dovolj dobro ali dovolj pogosto. Zato so za vodstvo bolj primerne ocene sprejetosti na podlagi dejanskega izvajanja procesa, ki ga do določene stopnje lahko razberemo iz dnevniških zapisov. Razen pogostosti izvajanja aktivnosti, ostalih karakteristik ocenjevanja sprejetosti ni bilo mogoče razbrati iz dnevniških zapisov. Zato ostajajo ocene sprejetosti na podlagi zaznavanja razvijalcev še vedno pomemben vir podatkov za evalvacijo.

Poglavje 5

Sklepne ugotovitve

V magistrskem delu smo razvili pristop za evalvacijo procesov razvoja programske opreme, ki za ocenjevanje poleg povratne informacije zaposlenih upošteva tudi sistemske zapise o izvajanju procesov. Razviti pristop vključuje področja evalvacije poslovnih procesov, analize dnevniških zapisov in orodij za nadzor različic (opisanih v poglavju 2). Pristop temelji na obstoječih evalvacijskih pristopih, pri katerih zaposleni ocenjujejo sociološki, tehnični in ekonomski vidik poslovnih procesov. Glavni doprinos našega pristopa je vpeljava analize dnevniških zapisov, ki ocenjevalcem služi kot dokaz o pogostosti in kakovosti izvajanja poslovnih procesov. Z dodatnim virom podatkov o procesih smo zmanjšali vpliv človeškega zaznavanja na ocenjevanje in povečali zaupanje vodstva v rezultate evalvacije.

V okviru pluralne študije primera smo preverili uporabnost razvitega pristopa v treh podjetjih. Z uporabo razvitega pristopa, ki je opisan v poglavju 3, smo najprej pridobili ocene razvojnih procesov od zaposlenih. Nato smo pridobljene ocene in meritve izvajanj procesov iz dnevniških zapisov posredovali tehnično naprednemu osebju, ki je ocenilo morebitna odstopanja med omenjenimi ocenami in meritvami. V primeru odstopanj so na podlagi meritev podali svojo oceno, ki je vplivala na končno oceno sprejetosti aktivnosti. Tehnično osebje je podalo oceno pogostosti izvajanja na podlagi meritev v približno 53 odstotkih primerov. S tem smo dosegli uporabo dnevniških

zapisov pri evalvaciji. Rezultate ocenjevanja smo posredovali vodstvom podjetij, ki so potrdila njihovo uporabnost za nadaljnje odločitve.

S preizkusom pristopa v praksi smo ugotovili, da so za vodstvo podjetij bolj primerne ocene pogostosti in kakovosti izvajanja procesov, ki temeljijo na meritvah izvajanja in zaznavanju razvijalcev kot tiste, ki temeljijo samo na zaznavanju. Dnevniški zapisi so se izkazali kot primeren vir takih meritev. Zaznavanje razvijalcev tudi v našem pristopu ostajajo ključen dejavnik pri ocenjevanju sociološkega vidika, saj je za izboljšanje sprejetosti pomembno razumeti razloge za trenutno stanje, ki so odvisni od zaznavanja razvijalcev.

Ena izmed večjih omejitev pri uporabi razvitega pristopa je časovna zahtevnost. Večina orodij razvojnega okolja nima enotne oblike dnevniških zapisov, kar poveča potreben čas za pripravo podatkov. Vse dnevniške zapise je treba preoblikovati v enotno obliko, ki jo lahko programi za analizo prepoznajo. Časovno zahtevno je tudi definiranje ustreznih meril za merjenje izvajanja bolj kompleksnih aktivnosti. V tem primeru se je treba poglobiti v povezanost in pomen dnevniških zapisov. Povečana časovna zahtevnost evalvacije zmanjšuje njeno koristnost iz ekonomskega vidika. Vprašanje, ki se pojavi kot posledica omejitve časovne zahtevnosti je, ali je mogoče izboljšati zaznano pogostost in kakovost izvajanja procesov do te mere, da z uporabo dnevniških zapisov v okviru vrednotenja ne bi pridobili nobene dodatne informacije. Za odgovor na to vprašanje bi lahko v prihodnosti pristop uporabili na primerih, kjer je delo manj fleksibilno, procesi pa bolj definirani in formalizirani. S tem bi zmanjšali vpliv napačnih interpretacij zahtevane pogostosti izvajanja, ki je bil prepoznan kot glavni vpliv na odstopanje zaznanih ocen od meritev v našem delu.

Iz tehnične plati bi bilo smiselno pristop preveriti na primerih z neomejenim dostopom do vseh dnevniških zapisov razvojnega okolja. Z dostopom do bolj podrobnih dnevniških zapisov orodja za nadzor izvorne kode bi lahko preverili uporabnost bolj kompleksnih meril, ki bi nam omogočila podroben vpogled v dejansko izvajanje procesov. Poleg poglobitve v podrobnosti dnevniških zapisov bi bilo smiselno analizirati dnevniške zapise več orodij in

s tem pridobiti boljši vpogled v povezanost večjega števila različnih poslovnih procesov. Pristop bi lahko v prihodnosti prilagodili tudi za vrednotenje sprejetosti in učinkovitosti drugih poslovnih procesov.

Literatura

[1] D. Abbott, "Chapter 15 - source code control—git," v *Linux for Embedded and Real-time Applications (Third Edition)*, third edition ed., ser. Embedded Technology, D. Abbott, Ed. Oxford: Newnes, 2013, str. 233 – 245.

[2] R. Agarwal in J. Prasad, "The role of innovation characteristics and perceived voluntariness in the acceptance of information technologies," *Decision Sciences*, zv. 28, št. 3, str. 557–582, 1997.

[3] I. Ajzen, *From Intentions to Actions: A Theory of Planned Behavior*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, str. 11–39.

[4] A. Alali, H. Kagdi, in J. I. Maletic, "What's a typical commit? a characterization of open source software repositories," v *2008 16th IEEE International Conference on Program Comprehension*, June 2008, str. 182–191.

[5] N. H. Anderson, "Foundations of information integration theory," 1981.

[6] R. Atkinson, "Project management: cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria," *International Journal of Project Management*, zv. 17, št. 6, str. 337 – 342, 1999.

[7] W. Bissi, A. G. S. S. Neto, in M. C. F. P. Emer, "The effects of test driven development on internal quality, external quality and productivity: A

systematic review," *Information and Software Technology*, zv. 74, št. Supplement C, str. 45 – 54, 2016.

[8] J. M. Bland in D. G. Altman, "Statistics notes: measurement error," *Bmj*, zv. 313, št. 7059, str. 744, 1996.

[9] W. Borman, "Performance evaluation in work settings," v *International Encyclopedia of the Social & Behavioral Sciences*, N. J. Smelser in P. B. Baltes, Eds. Oxford: Pergamon, 2001, str. 11 236 – 11 240.

[10] B. Chen, R. Curtmola, in J. Dai, "Chapter 17 - auditable version control systems in untrusted public clouds," v *Software Architecture for Big Data and the Cloud*, I. Mistrik, , R. Bahsoon, , N. Ali, , M. Heisel, , in B. Maxim, Eds. Boston: Morgan Kaufmann, 2017, str. 353 – 366.

[11] M. B. Chrissis, M. Konrad, in S. Shrum, *CMMI guidlines for process integration and product improvement*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[12] A. Cockburn, *Agile software development: the cooperative game*. Pearson Education, 2006.

[13] B. Collins-Sussman, B. Fitzpatrick, in M. Pilato, *Version control with subversion*. "O'Reilly Media, Inc.", 2004.

[14] G. Dalmarco, A. E. Maehler, M. Trevisan, in J. M. Schiavini, "The use of knowledge management practices by brazilian startup companies," *RAI Revista de Administração e Inovação*, zv. 14, št. 3, str. 226 – 234, 2017.

[15] H. K. Dam in A. Ghose, "Mining version histories for change impact analysis in business process model repositories," *Computers in Industry*, zv. 67, str. 72–85, 2015.

[16] W. Delone in E. Mclean, "Information systems success: The quest for the dependent variable. 1992," *Information Systems Research*, zv. 3, št. 1, 2010.

[17] G. K. Farber, "Can data repositories help find effective treatments for complex diseases?" *Progress in Neurobiology*, zv. 152, št. Supplement C, str. 200 – 212, 2017, developing drugs for neurological and psychiatric disorders: challenges and opportunities.

[18] W. Finlay, J. K. Martin, P. M. Roman, in T. C. Blum, "Organizational structure and job satisfaction: Do bureaucratic organizations produce more satisfied employees?" *Administration & Society*, zv. 27, št. 3, str. 427–450, 1995.

[19] M. Fishbein, "A theory of reasoned action: some applications and implications." 1979.

[20] D.-L. Florea, "The relationship between branding and diffusion of innovation: A systematic review," *Procedia Economics and Finance*, zv. 23, št. Supplement C, str. 1527 – 1534, 2015, 2nd GLOBAL CONFERENCE on BUSINESS, ECONOMICS, MANAGEMENT and TOURISM.

[21] T. Fritz, "Measuring individual productivity," v *Perspectives on Data Science for Software Engineering*, T. Menzies, L. Williams, in T. Zimmermann, Eds. Boston: Morgan Kaufmann, 2016, str. 67 – 71.

[22] M. J. Gallivan, "Organizational adoption and assimilation of complex technological innovations: development and application of a new framework," *ACM Sigmis Database*, zv. 32, št. 3, str. 51–85, 2001.

[23] "Archimate - orodje za modeliranje poslovnih komponent." Dostopno na: https://www.archimatetool.com/, (pridobljeno 29.7.2017).

[24] "Google forms - orodje za izdelavo anket." Dostopno na: https://www.google.com/forms/about/, (pridobljeno 25.8.2017).

[25] A. Giua in M. Silva, "Modeling, analysis and control of discrete event systems: a petri net perspective," *IFAC-PapersOnLine*, zv. 50, št. 1, str. 1772 – 1783, 2017.

[26] J. Han, J. Pei, in M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[27] P. Handayani, A. Hidayanto, A. Pinem, I. Hapsari, P. Sandhyaduhita, in I. Budi, "Acceptance model of a hospital information system," *International Journal of Medical Informatics*, zv. 99, št. Supplement C, str. 11 – 28, 2017.

[28] S. E. Harpe, "How to analyze likert and other rating scale data," *Currents in Pharmacy Teaching and Learning*, zv. 7, št. 6, str. 836 – 850, 2015.

[29] H. Huijgens, A. van Deursen, in R. van Solingen, "The effects of perceived value and stakeholder satisfaction on software project impact," *Information and Software Technology*, zv. 89, str. 19 – 36, 2017.

[30] E. Hustad, M. Haddara, in B. Kalvenes, "Erp and organizational misfits: An erp customization journey," *Procedia Computer Science*, zv. 100, št. Supplement C, str. 429 – 439, 2016, international Conference on ENTERprise Information Systems/International Conference on Project MANagement/International Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN / HCist 2016.

[31] P. Kruchten, *The rational unified process: an introduction*. Addison-Wesley Professional, 2004.

[32] D. Kumar in K. Mishra, "The impacts of test automation on software's cost, quality and time to market," *Procedia Computer Science*, zv. 79, št. Supplement C, str. 8 – 15, 2016, proceedings of International Conference on Communication, Computing and Virtualization (ICCCV) 2016.

[33] H. Lei, F. Ganjeizadeh, P. K. Jayachandran, in P. Ozcan, "A statistical analysis of the effects of scrum and kanban on software development projects," *Robotics and Computer-Integrated Manufacturing*, zv. 43, št. Supplement C, str. 59 – 67, 2017, special Issue: Extended Papers Selected from FAIM 2014.

[34] O. A. L. Lemos, F. F. Silveira, F. C. Ferrari, in A. Garcia, "The impact of software testing education on code reliability: An empirical assessment," *Journal of Systems and Software*, 2017.

[35] Z. Li, P. Avgeriou, in P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, zv. 101, št. Supplement C, str. 193 – 220, 2015.

[36] J. Loeliger in M. McCullough, *Version Control with Git: Powerful tools and techniques for collaborative software development.* "O'Reilly Media, Inc.", 2012.

[37] A. Maté, J. Trujillo, F. García, M. Serrano, in M. Piattini, "Empowering global software development with business intelligence," *Information and Software Technology*, zv. 76, str. 81–91, 2016.

[38] C. McCue, "Chapter 4 - process models for data mining and predictive analysis," v *Data Mining and Predictive Analysis (Second Edition)*, second edition ed., C. McCue, Ed. Boston: Butterworth-Heinemann, 2015, str. 51 – 74.

[39] M. Michlmayr, F. Hunt, in D. Probert, "Quality practices and problems in free software projects," v *Proceedings of the First International Conference on Open Source Systems*, 2005, str. 24–28.

[40] K. Mohan in F. Ahlemann, "Understanding acceptance of information system development and management methodologies by actual users: A review and assessment of existing literature," *International Journal of Information Management*, zv. 33, št. 5, str. 831–839, 2013.

[41] J. Park, J.-Y. Jung, in W. Jung, "The use of a process mining technique to characterize the work process of main control room crews: A feasibility study," *Reliability Engineering & System Safety*, zv. 154, str. 31–41, 2016.

[42] S. Park in Y. S. Kang, "A study of process mining-based business process innovation," *Procedia Computer Science*, zv. 91, št. Supplement C, str. 734 – 743, 2016, promoting Business Analytics and Quantitative Management of Technology: 4th International Conference on Information Technology and Quantitative Management (ITQM 2016).

[43] N. Paternoster, C. Giardino, M. Unterkalmsteiner, T. Gorschek, in P. Abrahamsson, "Software development in startup companies: A systematic mapping study," *Information and Software Technology*, zv. 56, št. 10, str. 1200 – 1218, 2014.

[44] J. C. Picken, "From startup to scalable enterprise: Laying the foundation," *Business Horizons*, zv. 60, št. 5, str. 587 – 595, 2017.

[45] "Promingit - orodje za oblikovanje in analizo dnevniških zapisov orodja git," Dostopno na: https://github.com/MilosLukic/promingit, (pridobljeno 15.10.2017).

[46] L. Raichelson, P. Soffer, in E. Verbeek, "Merging event logs: Combining granularity levels for process flow analysis," *Information Systems*, zv. 71, št. Supplement C, str. 211 – 227, 2017.

[47] C. K. Riemenschneider, B. C. Hardgrave, in F. D. Davis, "Explaining software developer acceptance of methodologies: a comparison of five theoretical models," *IEEE transactions on Software Engineering*, zv. 28, št. 12, str. 1135–1145, 2002.

[48] P. C. Rigby, A. Bacchelli, G. Gousios, in M. Mukadam, "Chapter 9 - a mixed methods approach to mining code review data: Examples and a study of multicommit reviews and pull requests," v *The Art and Science*

of Analyzing Software Data, C. Bird, T. Menzies, in T. Zimmermann, Eds. Boston: Morgan Kaufmann, 2015, str. 231 – 255.

[49] E. M. Rogers in F. F. Shoemaker, "Communication of innovations; a cross-cultural approach." 1971.

[50] E. Rojas, J. Munoz-Gama, M. Sepúlveda, in D. Capurro, "Process mining in healthcare: A literature review," *Journal of biomedical informatics*, zv. 61, str. 224–236, 2016.

[51] C. Romero in S. Ventura, "Educational data mining: A survey from 1995 to 2005," *Expert systems with applications*, zv. 33, št. 1, str. 135–146, 2007.

[52] V. Rubin, C. Günther, W. van der Aalst, E. Kindler, B. van Dongen, in W. Schäfer, "Process mining framework for software processes," *Software process dynamics and agility*, str. 169–181, 2007.

[53] G. A. Rummler in A. P. Brache, *Improving performance: How to manage the white space on the organization chart.* John Wiley & Sons, 2012.

[54] C.-w. Shen in C.-J. Kuo, "Learning in massive open online courses: Evidence from social media mining," *Computers in Human Behavior*, zv. 51, str. 568–577, 2015.

[55] "Slack - orodje za komunikacijo znotraj organizacije." Dostopno na: https://slack.com/, (pridobljeno 30.9.2017).

[56] X. Sun, B. Li, H. Leung, B. Li, in Y. Li, "Msr4sm: Using topic models to effectively mining software repositories for software maintenance tasks," *Information and Software Technology*, zv. 66, str. 1–12, 2015.

[57] S. Suriadi, R. Andrews, A. ter Hofstede, in M. Wynn, "Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs," *Information Systems*, zv. 64, št. Supplement C, str. 132 – 150, 2017.

[58] W. F. Tichy, "Rcs — a system for version control," *Software: Practice and Experience*, zv. 15, št. 7, str. 637–654, 1985.

[59] "Trello - orodje za upravljanje z nalogami," Dostopno na: https://trello.com/, (pridobljeno 30.9.2017).

[60] S. ur Rehman Toor in S. O. Ogunlana, "Beyond the 'iron triangle': Stakeholder perception of key performance indicators (kpis) for large-scale public sector development projects," *International Journal of Project Management*, zv. 28, št. 3, str. 228 – 236, 2010.

[61] J. L. Urda, B. Larouere, S. D. Verba, in J. S. Lynn, "Comparison of subjective and objective measures of office workers' sedentary time," *Preventive Medicine Reports*, zv. 8, št. Supplement C, str. 163 – 168, 2017.

[62] W. Van der Aalst in B. van Dongen, "Discovering workflow performance models from timed logs," *Engineering and Deployment of Cooperative Information Systems*, str. 107–110, 2002.

[63] W. M. van der Aalst, H. A. Reijers, A. J. Weijters, B. F. van Dongen, A. A. De Medeiros, M. Song, in H. Verbeek, "Business process mining: An industrial application," *Information Systems*, zv. 32, št. 5, str. 713–732, 2007.

[64] W. van der Aalst, B. van Dongen, J. Herbst, L. Maruster, G. Schimm, in A. Weijters, "Workflow mining: A survey of issues and approaches," *Data & Knowledge Engineering*, zv. 47, št. 2, str. 237 – 267, 2003.

[65] B. F. Van Dongen, A. K. A. de Medeiros, H. Verbeek, A. Weijters, in W. M. Van Der Aalst, "The prom framework: A new era in process mining tool support." v *ICATPN*, zv. 3536. Springer, 2005, str. 444–454.

[66] B. F. van Dongen in W. M. Van der Aalst, "A meta model for process mining data." *EMOI-INTEROP*, zv. 160, str. 30, 2005.

[67] D. Vavpotic in M. Bajec, "An approach for concurrent evaluation of technical and social aspects of software development methodologies," *Information and Software Technology*, zv. 51, št. 2, str. 528 – 545, 2009.

[68] D. Vavpotič in T. Hovelja, "Improving the evaluation of software development methodology adoption and its impact on enterprise performance," *Computer Science and Information Systems*, zv. 9, št. 1, str. 165–187, 2012.

[69] P. F. Velleman in L. Wilkinson, "Nominal, ordinal, interval, and ratio typologies are misleading," *The American Statistician*, zv. 47, št. 1, str. 65–72, 1993.

[70] V. Venkatesh in F. D. Davis, "A theoretical extension of the technology acceptance model: Four longitudinal field studies," *Management Science*, zv. 46, št. 2, str. 186–204, 2000.

[71] H. Verbeek, J. Buijs, B. van Dongen, in W. van der Aalst, "Xes tools," v *CAiSE Forum*, 2010, str. 1–8.

[72] A. J. Weijters in W. M. Van der Aalst, "Rediscovering workflow models from event-based data using little thumb," *Integrated Computer-Aided Engineering*, zv. 10, št. 2, str. 151–162, 2003.

[73] R. K. Yin, *Case study research: Design and methods*. Sage publications, 2013.

[74] Y. Yu, H. Wang, G. Yin, in T. Wang, "Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment?" *Information and Software Technology*, zv. 74, št. Supplement C, str. 204 – 218, 2016.

[75] T. Zimmermann, A. Zeller, P. Weissgerber, in S. Diehl, "Mining version histories to guide software changes," *IEEE Transactions on Software Engineering*, zv. 31, št. 6, str. 429–445, June 2005.