Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Maj Lenaršič

Model za ocenjevanje sprejetosti in zrelosti aktivnosti pristopa DevOps

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM
PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: izr. prof. dr. Damjan Vavpotič

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V okviru diplomske naloge pripravite model za ocenjevanje sprejetosti in zrelosti aktivnosti pristopa DevOps, ki bo pomagal vodjem in/ali procesnim inženirjem v podjetjih pri njegovem izboljševanju. Temeljite na obstoječih modelih za vrednotenje sprejetosti in zrelosti metodologij razvoja programske opreme, pri čemer upoštevajte specifike pristopa DevOps. Pripravljen model preizkusite v podjetju in ga na podlagi pridobljenih rezultatov kritično ovrednotite.

Iskreno se zahvaljujem družini, dekletu in prijateljem za vzpodbudo ob študiju ter mentorju, izr. prof. dr. Damjanu Vavpotiču, za strokovno pomoč in usmerjanje pri delu.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
CD	Continuous Deployment	Neprekinjena postavitev
CI	Continuous Integration	Neprekinjena integracija
CM	Configuration Management	Upravljanje konfiguracij
CMMI	Capability Maturity Model Integration	Poenoten zmožnostni zrelostni model
DOI	Diffusion of Innovations	Teorija difuzije inovacij
IaC	Infrastructure as Code	Infrastruktura kot koda
IEC	International Electrotechnical Commission	Mednarodna elektrotehniška komisija
ISO	International Organization for Standardization	Mednarodna organizacija za standardizacijo
IT	Information Technology	Informacijska tehnologija
MPCU	Model of Personal Computer Utilization	Model uporabe osebnega računalnika
PCI	Perceived Characteristics of Innovating	Zaznane karakteristike inovacij
SCM	Source control management	Upravljanje z izvorno kodo
TAM	Technology Acceptance Model	Model sprejetosti tehnologij
TPB	Theory of Planned Behaviour	Teorija načrtovanega vedenja
UI	User Interface	Uporabniški vmesnik
...	...	...

Povzetek

Naslov: Model za ocenjevanje sprejetosti in zrelosti aktivnosti pristopa DevOps

Avtor: Maj Lenaršič

Hitremu napredku programske opreme sledijo tudi metodologije razvoja. Pristop DevOps združuje nekatere sodobne filozofije, principe in prakse v eno takih metodologij. Orodij in aktivnosti, ki DevOps podpirajo, je vedno več, smiselnost njihove uvedbe pa se razlikuje od primera do primera. Izbira postaja vse zahtevnejša in odvisna od vse večjega števila faktorjev. Na osnovi že obstoječe relevantne literature smo razvili model za ocenjevanje trenutnega stanja DevOps metodologije v organizaciji. Model ocenjuje delovanje celotnega DevOps življenjskega cikla, s poudarkom na ocenjevanju sprejetosti posameznih orodij. Z izvedbo ocenjevanja podjetja pridobijo uvid v trenutno stopnjo zrelosti uvedene metodologije in ideje za izboljšanja stanja. Za ugotavljanje uporabnosti modela smo izvedli študijo primera v slovenskem podjetju.

Ključne besede: DevOps, programska oprema, metodologija, model, ocenjevanje.

Abstract

Title: Acceptance and maturity evaluation model for DevOps activities

Author: Maj Lenaršič

The software development process needs to keep up with the rapid growth of our software needs. DevOps methodology unites the newest philosophies, principles and practices of the field. The number of tools and activities that support these practices is growing and, with their usefulness depending on the specifics of the organization, choosing the right ones is becoming difficult. The companies have to analyse a growing number of factors. On the basis of existing literature we developed a model which evaluates the current level of DevOps in an organization. The model evaluates the whole development life cycle with the focus on acceptance of individual elements (tools, activities...). It will provide organizations with an overview of maturity of the methodology and a number of improvement scenarios. The model was tested with a case study in a Slovenian company.

Keywords: DevOps, software, methodology, model, evaluation.

Poglavje 1

Uvod

Osebje, ki skrbi za razvoj in vzdrževanje programske opreme znotraj podjetja se navadno deli v dve ekipi. Glavna naloga razvojne ekipe je hiter razvoj novih funkcionalnosti, aplikacij in sistemov. Ekipa za podporo delovanju pa skrbi za to, da so ti sistemi in aplikacije vedno delujoči, stabilni in varni. Taka organizacija v primeru delovanja obeh ekip v izolaciji lahko pelje v konflikt interesov, ki povsem onemogoči delovanje celotnega oddelka [26]. Vsaka sprememba vpeljana s strani razvijalcev ogrozi glavno nalogo ekipe za podporo delovanju. Metodologija za razvoj in vzdrževanje programske opreme DevOps je skupek principov, praks, metod in orodij, ki te meje med ekipama v večji meri zabrišejo.

Ocenjevanje smiselnosti in uspešnosti uvedbe DevOps elementov znotraj organizacije je zahteven problem. Pristopov je veliko, nove metode in orodja zahtevajo dodatno znanje in čas, smiselnost vpeljave posameznih elementov pa je odvisna od mnogo različnih faktorjev. Na ocenjevanje je treba pogledati iz vidika različnih vlog znotraj procesa razvoja. Odločili smo se za razdelitev osebja v dve skupini. V skupino izvajalcev spadajo razvijalci, testerji in ekipa za podporo delovanju, med procesne inženirje pa zaposleni, ki so najbolj odgovorni za oblikovanje procesov in delovanje celotnega oddelka. V klasičnih organizacijah so to navadno projektni vodje, vodje ekip in IT vodja podjetja.

V okviru diplomske naloge želimo torej podrobno pregledati področje

DevOpsa. Poizkusili bomo identificirati glavne prakse metodologije in prepoznati orodja, ki te neposredno podpirajo. Rezultat naloge bo model, ki podjetjem pomaga oceniti koristnost posameznih vpeljanih elementov in nivo zrelosti posameznih delov razvojno operativnega cikla. Razvrstitev elementov DevOps cikla bomo predstavili v dveh dimenzijah glede na mnenja ekip predstavljenih v prejšnjem odstavku. Model bomo preizkusili s študijo primera v slovenskem podjetju.

Diplomsko delo je sestavljeno iz šestih poglavji. Poleg uvoda in sklepne ugotovitve je vsebinska struktura dela naslednja. V 2. poglavju bomo podrobno pregledali literaturo DevOpsa in predstavili različne faze DevOps modela z orodji in aktivnostmi na katerih slonijo. Spoznali bomo tudi obstoječe modele in ogrodja za vrednotenje elementov znotraj procesa razvoja programske opreme. V 3. poglavju bomo predstavili izdelan model, njegovo strukturo in metodologijo izpeljave. V 4. poglavju pa bomo opisali izvedeno študijo primera v podjetju, predstavili rezultate ocenjevanje ter predstavili mnenje vodstva o modelu.

Poglavje 2

Pregled področja

V tem poglavju bomo predstavili najpomembnejšo literaturo za področje diplomske naloge. Predstavljen bo pristop DevOps kot metodologija razvoja programske opreme. Za temelj metodologije bomo opisali glavne principe ter predstavili najpomembnejše izboljšave, ki jih DevOps uvede v proces razvoja in upravljanja programske opreme. Metodologijo bomo nato razdelili na faze in opisali glavne prakse, aktivnosti in orodja, ki jih sestavljajo. Predstavili bomo še modele in teorije na katerih temelji naš model.

2.1 Kaj je DevOps?

DevOps je eden od trenutno najaktualnejših pristopov na področju razvoja programske opreme. Že iz imena se da razbrati, da poudarja potrebo po približevanju dveh navadno ločenih taborov. Razvijalcev (ang. *Dev - developers*) in ekipe za podporo operativnemu delovanju informacijskih tehnologij (ang. *Ops - operations* oziroma bolj specifično *IT operations*). Glede definicije izraza so viri pogosto neusklajeni. Pogosto se DevOps opiše le kot skupek sodobnih konceptov, praks, orodij in aktivnosti [14] za razvoj in vzdrževanje programske opreme, v nekaterih primerih pa kot konkretno metodologijo [23]. Večinoma so si usklajeni glede koristi, ki jih lahko DevOps prinese organizaciji. Ob uspešni uvedbi naj bi omogočil hitrejši in bolj dinamičen proces

razvoja in vzdrževanja programske opreme ter s tem hitro prilagajanje sodobnemu trgu in željam strank [26]. To se doseže z upoštevanjem principov, ki z zbliževanjem in brisanjem mej med ekipami znotraj IT oddelka odpravlja nepotrebno izgubo časa in virov.

2.1.1 Konflikt med razvijalci in ekipo za podporo delovanju

V organizacijah lahko zaradi nasprotujočih se ciljev razvijalcev in ekipe za podporo delovanju pride do kroničnega konflikta (ang. *chronic conflict*) [26] [22]. Naloga razvijalcev je prilagajanje sistemov in aplikacij željam trga in potezam konkurence. Redke nove funkcionalnosti se izkažejo za poslovno koristne [16], zato obstaja potreba po hitrem razvoju in menjavanju prioritet glede funkcionalnosti. Razvijalce torej vodi potreba po spremembah [22]. Obratno pa člani ekipe za podporo delovanju informacijskih tehnologij primarno skrbijo za stabilnost teh aplikacij. Strankam morajo omogočiti varno, zanesljivo in stabilno storitev. Vsak nov popravek ali dodana funkcionalnost lahko zahteva prilagoditev sistemov in infrastrukture, kar ogrozi njihove primarne cilje. Avtor knjige DevOps for Developers [22] navaja ta konflikt kot glavno vodilo v vedno večjo izoliranost taborov, kjer s časom postaja problem že osnovna komunikacija. Pri razvoju programske opreme gre navadno za kompleksne sisteme za katere je potrebno veliko specifičnega znanja, poznavanja različnih tehnologij in dobrega razumevanje želj in pričakovanj strank [17]. Na njih lahko delajo ogromne ekipe z različnimi organizacijskimi značilnostmi. Dober pretok znanja in informacij med sodelavci je bistvenega pomena, pomanjkanje tega pa lahko vodi do totalnega propada sistema.

Knjigi DevOps handbook [26] in The Phoenix Project [25] opišeta primer takega propada znotraj organizacij v fazah. Začne se z nabiranjem tehničnega dolga (ang. *technical debt*), kamor sodi slaba dokumentacija, pomanjkljiva infrastruktura in nepopolni ali celo neobstoječi testi. To so napake, ki dela sicer ne onemogočajo, a ga konstantno otežujejo in podaljšujejo. Reševanje teh težav se odlaga in delo na sistemih postaja počasno in neodzivno. Uva-

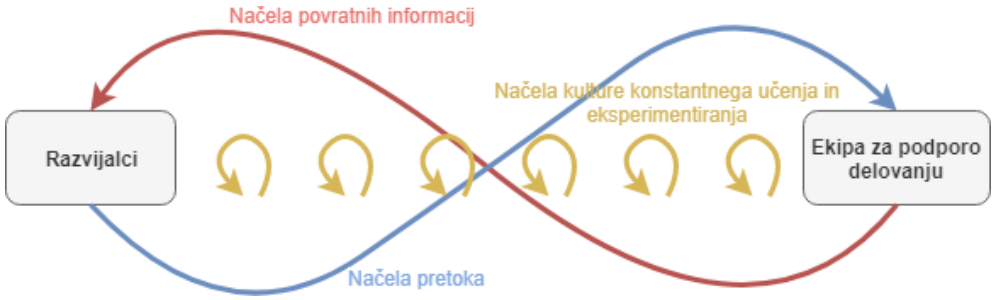
janje sprememb postaja vse bolj nestabilno in v sistemih se napake pojavljajo vse pogosteje. Vodstvo zaradi pomanjkljivega vpogleda v dogajanje sprejema nove projekte še preden se odpravijo stare napake. Na razvijalsko ekipo padejo nove odgovornosti. Pomanjkanje časa povzroči ogromno hitrih in površnih rešitev, tehnični dolg pa iz dneva v dan hitreje raste. Sledi zadnja faza. Zaposleni hitijo, menjajo med delom na novih projektih in popravljanjem starih napak. Delo čaka v dolgih čakalnih vrstah, kar upočasni njegovo predajo med različnimi ekipami. Potrebno je še več koordinacije in komunikacije, za katero pa sedaj ni več časa. Ozračje v oddelku postane bolj negativno, vzdušje je stresno, zaposleni pa vedno manj produktivni. Taka negativna spirala podaljša cikle razvoja. Povratne informacije strank, ki pomagajo pri planiranju naslednjih verzij izdelka, pa pridejo prepozno. Organizacija preneha dosegati zastavljene poslovne cilje in začenja vedno bolj zaostajati za konkurenco.

Principi DevOps se osredotočijo na reševanje kroničnega konflikta. Z izboljšanjem pretoka dela, razumevanja strank in gojenjem kulture stalnega eksperimentiranja in učenja, znižajo čas, potreben za izdajo novih funkcionalnosti. Vodilne organizacije na področju izdajajo nove spremembe v produkcijo tudi več kot desetkrat dnevno [26]. Stranka lahko tako preizkusi spremembe, ki jih je uvedel razvijalec še isti dan [34].

2.1.2 Principi DevOps

Gene Kim, avtor številnih knjig in eden od utemeljiteljev pristopa DevOps, nosilne principe razdeli v tri kategorije [26] grafično prikazane na sliki 2.1. Te principe bomo kasneje uporabili kot podlago za izpeljavo glavnih faz in praks razvojnega cikla DevOps.

Prva skupina principov se ukvarja s pretokom dela od razvijalcev do ekipe za podporo delovanju. Osredotočeni so na zmanjšanju časa, potrebnega za izdajo novih sprememb v produkcijsko okolje. Velik del hitrega razvojnega cikla je majhno število ročnih predaj med ekipami. To se doseže z avtomatizacijo in s primerno ureditvijo ekip. Z omejevanjem velikosti serij in



Slika 2.1: Tri skupine glavnih principov DevOps metodologije.

dela v teku (ang. *work in process*) se poskrbi za hitrejše reagiranje na nepredvidene težave. Tu se DevOps zgleduje po principih, definiranih znotraj vitke proizvodnje [24]. Namesto, da se za izdajo po cevovodu pošilja po 10 sprememb hkrati, se v trenutku vsake spremembe sproži cevovod, ki nato nove funkcionalnosti kot po tekočem traku v proizvodnji pošlje v naslednje faze razvojnega cikla. To omogoča, da se istih napak ne ponavlja večkrat, saj se potencialne težave v kasnejših fazah opazi takoj. Predvsem pa nove funkcionalnosti in spremembe do strank pridejo veliko hitreje. Posledično se poveča zadovoljstvo strank in prejme povratne informacije pravočasno, kar omogoča ustrezno prilagajanje naslednje iteracije sprememb.

Za uspešno uporabo uporabo teh informacij in ustrezno prilagajanje naslednjega cikla skrbi druga skupina principov povratnih informacij. Stare slapovne metodologije so konkretne povratne informacije strank omogočale šele ob končni izdaji izdelka. S hitrejšimi iteracijami delovnega procesa in uvedbo telemetrije se na vsakem nivoju procesa težave zazna ob njihovem nastanku. DevOps poudarja tudi hitro odpravljanje težav s sistemom, ki se zgleduje po konceptu Andon kabla (ang. *Andon cord*) iz vitke proizvodnje [2]. Ob pojavitvi problema postane odpravljanje tega problema glavna prioriteta celotnega oddelka. Delo se normalno nadaljuje šele po uspešni odpravi. Premikanje nadzora kakovosti v procesu bližje izvirom težav zagotovi, da za težave ni odgovoren le en ločen oddelek (v drugih metodologijah navadno ekipa za zagotavljanje kakovosti). Razvijalec sam lahko s sprožitvijo

avtomatiziranih testov skrbi za preverjanje varnosti aplikacije. S sledenjem natančno definiranih nefunkcionalnih zahtev pa optimizira svojo spremembo za naslednjo ekipo. S tem se poskrbi za izboljšanje predaj dela med ekipami, kar omogoča boljšo klimo celotnega oddelka.

S samo klimo oziroma kulturo znotraj organizacije se ukvarja tretja skupina načel. Kim poudarja pomembnost kulture konstantnega učenja in eksperimentiranja znotraj organizacij. Ob pojavu napak naj se ne išče krivcev, ampak načine izogibanja tem situacijam v prihodnosti. Pomembno je stalno izboljševanje delovnega procesa. Eksperimentira se lahko z vpeljavo kontroliranih stresnih testov. Vsako novo odkritje se dokumentira in deli s celotnim oddelkom. Rezultat vpeljave teh načel je večja zanesljivost sistemov ter večji občutek varnosti in pripadnosti med zaposlenimi.

2.2 Razvojno operativni cikel DevOps

Vse glavne prakse in aktivnosti povezane z načinom razvoja DevOps lahko povežemo s principi, opisanimi v prejšnjem poglavju. Ključne prakse DevOps, kot so neprekinjena integracija kode (CI), neprekinjena postavitev, neprekinjene izdaja in uporaba arhitekture mikrostoritev večinoma izhajajo iz principov pretoka. Principe povratnih informacij opazimo pri praksah stalnega spremljanja zdravja in obremenitve sistemov (ang. *monitoring*) ter shranjevanja in analize dnevnikov (ang. *logging*). Principe kulture pa se opazi pri uporabi orodij za deljenje znanj med sodelavci in komunikacijo, ki presegajo meje ene ekipe. V tem poglavju bomo s pomočjo različnih virov proces DevOps razdelili na različne faze. Opisali bomo glavne značilnosti vsake, vključno s pripadajočimi praksami, aktivnostmi in orodji.

Za uspešno vpeljavo filozofije DevOps v organizacijo ni natančno določenih smernic. Za različne primere podjetji so primerne različne topologije ekip, različna orodja in različne aktivnosti [6]. Viri si tudi glede same razdelitve življenjskega cikla DevOps niso povsem enotni. Primer razdelitve cikla na faze je prikazan na sliki 2.2. V članku Wahaballa et al. [43], predlagajo poe-

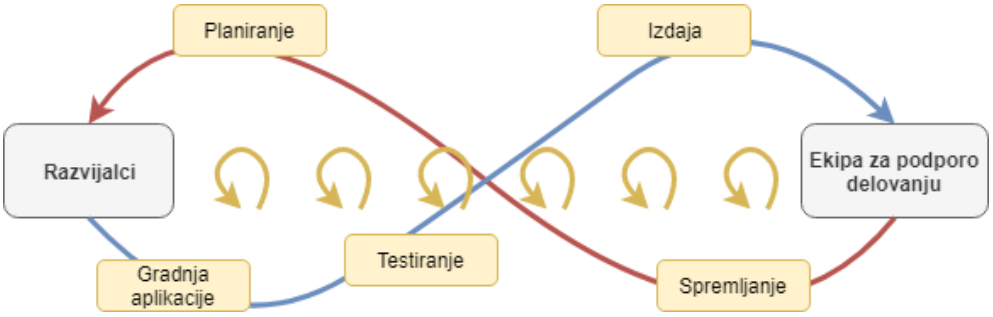
noten model DevOps (ang. *Unified DevOps Model*). Življenjski cikel DevOps predstavijo v štirih delih in za vsakega definirajo potek aktivnosti:

- V fazo planiranja (ang. *Plan*) sodi pregled obveznosti, ocena potrebnih virov, prioritizacija in načrtovanje naslednje iteracije.
- Naslednja faza je razvoj (ang. *Develop*). Sem sodi razvoj novih funkcionalnosti, njihova dokumentacija in pregled kode za skupinsko odpravljanje napak.
- Sledi faza testiranja (ang. *Testing*), kamor spada nastavljanje testnih okolij, identifikacija strategije testiranja in sama detekcija napak oziroma pomanjkljivosti v aplikaciji.
- Faza neprekinjene izdaje (ang. *Continuous Release*) pa opiše aktivnosti, katerih cilj je planiranje, upravljanje in izvedba urnika izdaj.

Amazon [14] model DevOps razdeli na pet delov in sicer na gradnjo (ang. *Build*), testiranje (ang. *Test*), izdajo (ang. *Release*), nadzorovanje (ang. *Monitor*) in planiranje (ang. *Plan*). Cikel predstavijo kot tok vrednot med podjetjem in njihovimi strankami. Drugi viri v ospredje postavljajo predvsem tok med razvijalci in ekipo za podporo delovanju [26, 36]. Meje med fazami in različnimi praksami so precej zamegljene ter natančna razdelitev ostaja nedefinirana. Za naše potrebe razdelitve se bomo zgledovali po navedenih virih.

2.2.1 Razvoj

DevOps se glede načina razvoja programske opreme opira na načela agilnega razvoja programske opreme. Poudarek se daje na zadovoljstvu strank, čim večji enostavnosti in organizaciji samozadostnih multifunkcijskih ekip razvijalcev. To organizacijam omogoči hitro izdajanje novih izdelkov oziroma posodobitev v natančno določenih intervalih [15].



Slika 2.2: Ena od možnih razdelitev cikla DevOps na faze.

V zvezi z razvojem se pogosto omenja uporabo arhitekture mikrostoritev z virtualizacijo na nivoju operacijskega sistema (ang. *containerization*) in sodelovanje [18]. Za delo z vsebniki je najbolj razširjen odprtokodni Docker [28], pri sodelovanju pa se pogosto omenjajo prakse pregleda kode med sodelavci (ang. *Code review*) in pomembnost ekvivalentnosti testnih in produkcijskih okoljih, kar prinese dobro sodelovanje z operativnim delom organizacije [31].

Značilnost metodologije DevOps je tudi drugačen pogled na zagotavljanje kakovosti programske opreme. To ni več delo specializirane ekipe, ampak glavno vodilo celotnega oddelka. Nadzor kakovosti se izvaja na vsakem koraku zanj po so odgovorni vsi deležniki razvojno operativnega cikla. Razvijalec vsako potrjeno spremembo izvorne kode (sistemi za upravljanje z izvorno kodo so nenadomestljivi) požene skozi cevovod, ki vsebuje avtomatizirano testiranje. V bolj zrelih okoljih DevOps se aplikacija avtomatsko namesti v testno okolje. Razvijalec tako lahko preveri kakovost svojega dela pred predajo dela naslednji ekipi oziroma delovni postaji [22].

2.2.2 Neprekinjena integracija in testiranje

Zgoraj omenjeni proces je del ene od temeljnih praks, ki DevOps sestavljajo, neprekinjene integracije (ang. *Continuous Integration*) - CI. CI je praksa razvoja programske opreme, kjer se vsaka sprememba programske kode avtomatsko integrira s centralnim repozitorijem. Integracija novosti tako postane

povsem avtomatiziran proces, ki se ga nikoli ne preskoči. Vsaka sprememba se preveri z gradnjo programske rešitve ter testiranjem. Ta pristop ekipam omogoča hitrejše razvijanje novih funkcionalnosti, saj večina napak ostane lokalnih in so odpravljene ob njihovem nastanku [20].

Za ta proces skrbijo specializirana orodja. Strežniki, ki omogočajo integracijo z drugimi orodji za gradnjo, testiranje in dostavljanje artefaktov programske opreme. S pomočjo teh se izdela cevovode, ki tečejo na strežnikih in vsebujejo ukaze za izvedbo korakov integracije. Ob ustrezni vpeljavi so ta orodja povsem integrirana z načinom dela ekipe [38]. O napredku integracije je na primer razvijalec lahko obveščen ob trenutku spremembe stanja preko sporočilnih sistemov ali elektronske pošte. Primera takih orodij sta odprtokodni Jenkins [7] in Atlassian-ov Bamboo [3].

V cevovode po potrebi torej vključujemo različna druga orodja. Za orodje Jenkins je na voljo več kot 1000 različnih vtičnikov, ki omogočajo integracijo z najrazličnejšimi orodji [8]. Prevladujejo vtičniki za gradnjo aplikacij (na primer Maven in Gradle) in orodja za testiranje.

Vrst testiranja je več, možnosti glede izbire orodij pa ogromno. V knjigi DevOps for Developers [22] avtor izpostavi pomembnost stalnega avtomatskega testiranja. Razdeli ga v 4 kategorije:

- **Testiranje enot** (ang. *Unit Testing*) - Testirajo se posamezne komponente programske opreme. Tako se odvzame odvisnosti do drugih virov, podsistemov in drugih komponent. Orodja iz družine xUnit predstavljajo standard tega področja in so navadno dobro podprta s strani strežnikov za neprekinjeno integracijo.

- **Testiranje storitev** (ang. *Service Testing*) - Storitve so sestavljene iz večih enot. Ta vrsta testiranja je pogosto imenovana integracijsko testiranje. S tem testiramo povezanost in interakcijo različnih komponent. Dober primer orodja za to vrsto testiranja je Arquillian [40].

- **Testiranje uporabniškega vmesnika** (ang. *UI Testing*) - Testira se celoten sistem, torej aplikacija, vmesni sistemi in infrastruktura. Pisa-

nje testov za uporabniški vmesnik je zahtevno in zahteva veliko časa, zato morajo biti enostavni in dovolj robustni, da so prizanesljivi do manjših prilagoditev uporabniškega vmesnika. Primer orodja za testiranje uporabniškega vmesnika je Selenium.

- **Statična analiza kode** (ang. *Static Code Analysis*) - V nasprotju z drugimi vrstami testiranja se statično analizira kodo še pred izgradnjo programa. Uporablja se podobne tehnike kot pri prevajanju kode. Razširjen primer takega orodja je SonarQube [13].

2.2.3 Izdaja

Za izdajo programske opreme se v okoljih DevOps najpogosteje omenja praksi neprekinjena dostava (ang. *Continuous Delivery*) in neprekinjena postavitev (*Continuous Deployment*). Cilj je avtomatizirati proces dostave in izdaje programske opreme v različna okolja. Obstaja več šol mišljenja za omenjeni praksi. Nekatere jih definirajo kot podaljšek CI prakse in pojma velikokrat uporabljajo izmenično. Drugi razliko vidijo v avtomatizaciji izdaje sprememb v produkcijsko okolje. Neprekinjena dostava se zanaša na ročno izdajo kode med končne uporabnika. V praksi neprekinjene postavitve pa se tudi ta korak avtomatizira [30].

Zrele organizacije, ki se metodologije DevOps posložujejo, s pomočjo teh praks nove funkcionalnosti izdajajo tedensko ali celo dnevno. Večina orodij na področju se kvalificira kot CI/CD orodja, torej so sposobna izvedbe celotnega cevovoda. Tega procesa sta ne primer zmožna že prej omenjena Jenkins in Bamboo. Obstajajo tudi nekoliko bolj specializirana orodja za izdajanje. IBMov UrbanCode je orodje namenjeno striktno upravljanju in avtomatizaciji izdaj [29].

2.2.4 Upravljanje

Operativni del ekip ali organizacij skrbi za upravljanje sprememb, aplikacij, strežnikov in konfiguracij. V tem delu cikla DevOps se poudarja praksa

infrastruktura kot koda (ang. *Infrastructure as Code*). IaC je avtomatizacija IT operacij s pomočjo kode. Zagovarja uprabljanje z infrastrukturo s pomočjo podobnih praks in tehnologij kot za razvoj programskih rešitev. Uporablja se SCM sisteme in CI/CD prakso. S tem se standardizira upravljanje z infrastrukturo in strežniki, kar omogoča večjo ponovljivost in transparentnost vsake sprememb [14].

Za navedeno skrbijo različna orodja. Orodja za upravljanje konfiguracij (ang. *configuration management*) omogočajo uporabo posebnih deklarativnih jezikov za opisovanja sistemski konfiguracij ter s tem odpravijo potrebo po ročnih posegih v strežnike in infrastrukturo. Repliciranje stanja postane enostavnejše s popolnim vpogledom zgodovine posegov v konfiguracije [19]. Primeri takšnih orodij so Ansible, Puppet in Salt [37].

Druga kategorija orodij, ki so prisotna v tej fazi, so orodja za orkestracijo konfiguracij. Za razliko od orodij za upravljanje konfiguracij se ta primarno ne uporablja za nastavljanje konfiguracij različnih sistemov in programske opreme, ampak za dodajanje novih sistemov in usklajevanje celotne infrastrukture. Omogočajo hitro prilagajanje arhitekture strojne opreme, na katerih programska oprema sloni in prilagajanje spremembam poslovnih potreb. Predstavnik te skupine orodij je Terraform [35].

2.2.5 Spremljanje

Za pomoč pri sprejemanju odločitev pri razvoju ali izdaji programskih rešitev ter upravljanju infrastrukture se v DevOps zelo razširjeno uporablja orodja za nadzor sistemov, dnevnikov delovanja in aplikacij (ang. *monitoring*). Taka orodja zagotavljajo pravočasno detekcijo napak in problemov ter spremljanje celotnega razvojno operativnega procesa [18]. V okolju DevOps je vsak član IT oddelka zadolžen za spremljanje [26]. Poudarja se pomembnost konsolidacije in ustrezne kategorizacije teh podatkov in informacij. Orodja za nadzor se okvirno deli v tri kategorije [5]:

- **Nadzor infrastrukture** (ang. *Infrastructure Monitoring*). Primera orodij sta Nagios in Icinga.

- **Nadzor aplikacij** (ang. *Application Monitoring*). Pogosto sta uporabljena AppDynamics in New Relic.
- **Upravljanje zapisov v dnevnikih** (ang. *Log Management*). Tu je razširjen ELK stack.

2.2.6 Planiranje

Namen te faze je pregled in ocena vseh metrik in drugih povratnih informacij, ki smo jih zbrali v minulem razvojnem ciklu in preko orodij za spremljanje. Na podlagi teh se nastavi prioritete, oceni vire in sestavi načrt za naslednjo iteracijo. Poslovne potrebe se spremeni v zahtevke, ki se dodelijo razvijalcem v sistemih za nadzor zahtevkov (tu je razširjena Atlassian-ova Jira) [18]. Te vsebujejo različne sisteme za prioritizacijo in razporejanje zahtevkov in beleženje napredka (na primer implementacijo Kanban table). V fazo planiranja se velikokrat vključuje tudi orodja za deljenje znanj (Atlassion-ova rešitev Confluence [4], Slite [12],...) ter sporočilni sistemi (Slack, Microsoft Teams).

Razvojno operativni cikel se tako zaključi in ponovno začne. S konstantno iteracijo in postopnim izboljševanjem procesa postaja bolj avtomatiziran, varen in enostaven. S tem imajo programske rešitve aktualne funkcionalnosti ob pravem času, hkrati pa zadostujejo vsem standardom varnosti in delovanja.

2.3 Modeli za ocenjevanje procesa razvoja programske opreme

V tem delu bomo predstavili vire, ki raziskujejo in predstavljajo modele, ogrodja in načine ocenjevanja metodologij razvoja programske opreme. Osredotočili se bomo na modele, ki ocenjujejo stopnjo sprejetosti inovacij procesa in na modele, ki ocenjujejo zrelost procesov. Prvi dajejo poudarek na sociološki vidik sprejetja metodologij, slednji pa predvsem na tehnični vidik.

2.3.1 Ocenjevanje sprejetosti inovacij

Ogrodja in modeli, ki vrednotijo stopnjo sprejetosti inovacij v procesu razvoja programske opreme te obravnavajo iz sociooloških, kulturnih in psiholoških vidikov. Vrednotijo lahko celotne metodologije razvoja programske opreme ali pa njihove elemente posamično [41]. Pet primerov takšnih ogrodji je opisanih v članku Riemenschneider et al. [32]. In sicer Model sprejetosti tehnologij - TAM, njegova nadgradnja - TAM2, Zaznane karakteristike inovacij - PCI, Teorija načrtovanega vedenja - TPB in Model uporabe osebnega računalnika - MPCU. Večina takih modelov in ogrodji pa temelji na Rogersovi teoriji difuzije inovacij - DOI [33].

Rogers raziskuje razloge za uspešnost oziroma neuspešnost širitve inovacij in idej med končnimi uporabniki. Difuzija je proces širitve inovacij med člani določenega socialnega sistema. Njeni štirje glavni elementi pa so inovacija, komunikacijski kanali, čas ter socialni sistem. Ti elementi naj bi torej odločilno vplivali na to, ali bo določena nova tehnologija, praksa ali ideja sprejeta. Identificira pet glavnih karakteristik inovacij:

- **relativna prednost** (ang. *relative advantage*) - predstavlja prednost pred dosedanjimi praksami in idejami,
- **združljivost** (ang. *Compatibility*) - predstavlja skladnost z obstoječimi praksami in načinom dela,
- **zapletenost** (ang. *Complexity*) - predstavlja zapletenost uporabe in vpeljave,
- **možnost preizkušanja** (ang. *Triability*)) - predstavlja možnost, da uporabnik inovacijo lahko preizkusi,
- **možnost opazovanja** (ang. *Observability*) - predstavlja možnost, da uporabnik učinek inovacije lahko vidi pri drugih.

V članku Vavpotič et al.[41] je predstavljen primer modela ocenjevanja, ki za ocenjevanje iz sociološkega vidika črpa iz zgoraj omenjenih ogrodij in DOI

teorije. Skupina vprašanj izhaja prav iz navedenih karakteristik. Vprašalnik vsebuje vprašanja za evalvacijo relativne prednosti, združljivosti in zapletenosti elementov metodologij. Ta skupina vprašanj ocenjuje razloge za trenutni nivo sprejetosti elementa. Model ocenjuje elemente tudi po tehničnem vidiku, rezultate pa predstavi v razsevnem diagramu, ki ga razdeli v štiri kvadrante. Te razdelijo ocenjene elemente v štiri tipe glede na povprečno vrednost njihovih ocen. V vseh štirih primerih izvedbe študije primera v organizacijah za razvoj programske opreme je njihovo vodstvo potrdilo koristnost modela. Čez čas je bil nadgrajen še z ekonomskim vidikom [42]. Tudi v tem primeru so študije primerov potrdile pozitivne učinke uporabe modela.

2.3.2 Ocenjevanje zrelosti procesov

Eno od področji ocenjevanja, ki se ukvarja z ocenjevanjem metodologij s tehničnega vidika, so modeli in ogrodja za ocenjevanje zrelosti in učinkovitosti metodologij. Imenujemo jih zrelostni modeli. Zrelostni modeli lahko ocenjujejo ljudi, kulturo ali tehnologije [10]. V primeru ocenjevanja metodologij razvoja programske opreme pa so uporabljeni predvsem za ocenjevanje zrelosti procesov. Zrelost se v tem primeru nanaša na stopnjo formalnosti in optimiziranosti procesov. Primeri takih modelov oziroma standardov so ISO/IEC standardi in pa poenoten zmožnostno zrelostni model - CMMI [39].

CMMI je uporabljen za oceno procesov določenega projekta, divizije ali celotne organizacije. Poudarja pomembnost uporabe metodologije in dobro definiranih procesov. Opredeli pet zrelostnih stopenj, ki se lahko uporabijo kot merilo za izboljševanje procesov:

1. **začetna** (ang. *Initial*),
2. **upravljalna** (ang. *Managed*),
3. **opredeljena** (ang. *Defined*),
4. **merljiva** (ang. *Quantitatively managed*),
5. **optimizirana** (ang. *Optimization*).

Za začetne stopnje velja da so procesi še nepredvidljivi, slabo kontrolirani in reaktivni. Procesi skozi stopnje dosegajo vse večje standarde ter so bolj definirani, merljivi in nadzorovani. Za zadnjo stopnjo je njihova zrelost zadostna. Poudarek je nato le še na stalni optimizaciji [39].

Splošno prepričanje je, da CMMI deluje v nasprotju z agilnimi pristopi razvoja programske opreme. CMMI so v začetku uporabljali predvsem razvijalci obsežnih in kritičnih sistemov v organizacijah s strogo hierarhično strukturo in slapovno metodologijo razvoja programske opreme. To naj bi bil eden od razlogov za sloves slabe kompatibilnosti s sodobnimi načini razvoja. V članku [21] avtorji opišejo razloge za neupravičenost tega slovesa in možnosti uporabe obeh znotraj istega oddelka.

Primer integracije CMMI-ja z metodologijo DevOps predstavi Mohamed [27]. V svojem članku opredeli zrelostni model za DevOps, ki izhaja iz CMMI-ja. Ima torej 5 stopenj, vsaka pa se ocenjuje glede na štiri dimezije. Prva je kvaliteta (ang. *Quality*), ki omogoča dostavo programske opreme na agilen in hiter način z omejevanjem časa za predelovanje, ustvarjanjem stabilnih aplikacij in izdajami, ki dosegajo višje standarde kvalitete. DevOps karakteristika avtomatizacije (ang. *Automatization*) z avtomatizacijo procesov izboljša produktivnost, hitrost in ponovljivost. Karakteristika sodelovanja (ang. *Collaboration*) poudarja boljšo komunikacijo med različnimi ekipami z rabo različnih pristopov in orodij. Zadnja karakteristika pa je plast obvladovanja informatike (ang. *Governance*), ki je odgovorna za usklajenost vseh ostalih karakteristik in usmerjenost k doseganju glavnih ciljev metodologije DevOps in organizacije na splošno.

S tem modelom želi omogočiti organizacijam lažjo in uspešnejšo transformacijo življenjskega cikla razvoja programske opreme in hitrejši prevzem ključnih načel DevOps.

Poglavje 3

Razvoj modela

Na podlagi teoretične podlage opisane v prejšnjem poglavju smo ustvarili model za ocenjevanje metodologije DevOps, predstavljen v tem poglavju. Najprej bomo na kratko predstavili in povzeli celotni model. Nato se bomo poglobili v oba glavna vidika ocenjevanja modela. Sledil bo opis prikaza rezultatov modela. Poglavje pa bomo zaključili z opisom vseh korakov za izvedbo ocenjevanja.

3.1 Osnove

Naš model ocenjuje ustreznost metodologije DevOps znotraj organizacije. Vidika ocenjevanja sta zastavljena tako, da se po izvedbi modela trenutno stanje vpeljave metodologije oceni glede na tehnični in glede na sociološki vidik. Pred izvedbo modela osebje IT oddelka organizacije razdelimo v dve skupini. Člane prve skupine poimenujemo procesni inženirji. Mednje navadno spadajo vodja IT oddelka, vsi projektni vodje in vodje ekip. To so člani, ki naj bi poznali celotni razvojno operativni proces oddelka in so ključni pri izbiranju in oblikovanju procesov, ki so v metodologiji prisotni. V drugo skupino spadajo ostali deležniki razvojno operativnega cikla kot so razvijalci, preizkuševalci, člani ekipe za podporo delovanju, sistemski inženirji itn. Člane te skupine za potrebe modela imenujemo izvajalci.

Prvi vidik ocenjevanje ustreznosti metodologije DevOps neke organizacije je ocenjevanje sprejetosti. V tem delu DevOps razdelimo na elemente, ki so bodisi aktivnost, orodje ali kombinacija obeh. To nam omogoči identifikacijo specifičnih problematičnih delov metodologije. Spremljamo splošno sprejetost elementov in razliko med ocenami sprejetosti elementov obeh skupin. Glede na to lažje določimo možne ukrepe za izboljšanje sprejetosti aktivnosti in orodij že vpeljanih v razvojno operativni cikel. Pri ocenjevanju tehničnih vidikov metodologije ne obravnavamo kot skupek elementov, ampak kot zaporedje faz oziroma obdobij. Razdelitev na obdobja se zaradi čim večje jasnosti vprašanj zgodi v sodelovanju z organizacijo. Pri ocenjevanju tega vidika sodeluje le skupina procesnih inženirjev. S tem zagotovimo zadostno poznavanje širše slike procesa razvoja in vzdrževanja programskih rešitev.

Rezultate ocenjevanja predstavimo z diagramom in tabelami, ki omogočajo hiter pregled povprečnih vrednosti in razvrstitve elementov. Za problematične elemente izpostavimo zanimive porazdelitve odgovorov ali druga statistična odstopanja, ki nam pomagajo pri prepoznavanju ukrepov za izboljšanje stanja. Rezultate se po analizi rezultatov predstavi članu organizacije, ki je odgovoren za vpeljavo in delovanje metodologije DevOps.

3.2 Vidiki ocenjevanja

V tem podpoglavju bolj natančno opišemo zgoraj omenjena vidika ocenjevanja modela. Za oba vidika se podrobneje poglobimo v glavne karakterstike, ki ju sestavljajo.

3.2.1 Ocenjevanje sprejetosti elementov DevOps

Pri ocenjevanju sprejetosti ocenjujemo metodologijo DevOps po sociooloških vidikih in pogostosti uporabe ob priložnosti. V tem primeru metodologijo DevOps obravnavamo kot skupek orodij in aktivnosti, ki jih za potrebe našega modela poimenujemo elementi. S tem identificiramo sprejemanje orodij in aktivnosti DevOpsa, ki jih je organizacija v cikel razvoja že uvedla. Rezul-

tati tega vidika ocenjevanja organizacijam omogoči hiter pregled splošnega sprejetja elementov DevOps, ugotavljanje ukrepov in prioritizacijo naslednjih ukrepov za izboljšanje. Ocenjevanje sprejetosti se zgleduje po ogrodjih, opisanih v poglavju 2, karakteristike ocenjevanja pa v večini izhajajo iz dela [33]. Karakteristike sprejetosti elementov DevOps so:

- **Relativna prednost** - ocenjujemo stopnjo uporabnikovega dojemanja elementa kot boljšega od predhodnih rešitev. Je ena od karakteristik inovacij definiranih v Rogersovi teoriji DOI. Tipov relativne prednosti, ki jo lahko nek element nudi, je več. V našem modelu se posvetimo predvsem uporabnikovem mnenju glede izboljšanja produktivnosti in kvalitete dela.

- **Kompatibilnost** - s to karakteristiko se ocenjuje stopnjo uporabnikovega dojemanja elementa kot skladnega z njegovimi vrednotami, izkušnjami in potrebami. Izhaja iz Rogersove teorije DOI. Naš model s to karakteristiko ocenjuje sociološko kompatibilnost in skladnost.

- **Kompleksnost** - ocenjuje stopnjo uporabnikovega dojemanja elementa kot zahtevnega za uporabo in razumevanje. Tudi ta karakteristika izhaja iz Rogersove teorije DOI in je bila uporabljena v modelih in ogrodjih, opisanih v poglavju 2.

- **Pogostost uporabe ob priložnosti** - ocenjujemo pogostost dejanske uporabe elementa ob pojavitvi priložnosti. Izvajalci se uporabi elementov, ki jih slabo sprejemajo velikokrat poskusijo izogniti. Karakteristika je povzeta po člankih [41, 42].

Za ugotavljanje sprejetosti elementov bodo izvajalci in procesni inženirji odgovarjali na vprašanja, ki bodo ocenjevala sprejetost glede na zgornje karakteristike. Na vprašanja karakteristik relativna prednost, kompatibilnost in kompleksnost bodo izvajalci odgovarjali iz osebnega vidika, procesni inženirji pa iz vidika njihove celotne ekipe. Dodatno bodo izvajalci odgovarjali na vprašanje vezano na karakteristiko Pogostost uporabe ob priložnosti.

Vprašalnik izvajalcev je prikazan v tabeli 3.1, vprašalnik procesnih inženirjev pa v tabeli 3.2.

3.2.2 Ocenjevanje zrelosti faz DevOps

Za ocenjevanje tehnične ustreznosti metodologije DevOps procesni inženirji organizacije ocenjujejo zrelost metodologije glede na štiri glavne karakterstike metodologije DevOps. DevOps v tem primeru v sodelovanju z organizacijo razdelimo na posamezne faze. To nam omogoča lažjo identifikacijo delov razvojno operativnega cikla, kjer principi DevOps še niso zadostno prevzeti. Karakterstike izvirajo iz članka [27], kjer so predstavljene kot ključne dimenzije za uspešno sprejetje metodologije DevOps in prakse neprekinjene dostave. Vprašanja, ki se na njih nanašajo, pa so zastavljena tako, da mora osebje raven posamezne karakteristike primerjati s sorodnimi organizacijami oziroma njihovimi IT oddelki. Karakteristike zrelosti faz so:

- **Avtomatizacija** - raven avtomatizacije se ocenjuje glede na količino avtomatizacije in ustrezno dokumentacijo. Preverja se zrelost infrastrukture in raven tehničnega znanja osebja, ki se nanaša na poznavanje procesov in orodij, ki avtomatizacijo omogočajo. Z višanjem ravni avtomatizacije organizacije povečujejo hitrost dostave programske opreme, produktivnost ekip in ponovljivost procesov.

- **Sodelovanje** - ocenjevanje ravni sodelovanja daje poudarek na ustreznih procesih za komunikacijo znotraj organizacije. Preverja se ustrezna koordinacija ekip, definiranost vlog in odgovornosti, količina medekipne komunikacije, načine in procese sprejemanja odločitev, deljenje znanja med sodelavci ter obstoj ustreznih metrik komunikacijskih kanalov. Visoka zrelost karakteristike sodelovanja omogoča usklajenost ekip, minimiziranje tveganj in boljše razumevanje tehničnih zahtev.

- **Kvaliteta** - raven zagotavljanja kvalitete ocenjuje natančnost definiranih standardov na ravni organizacije, odkrivanje napak v programski

opremi s specializiranimi orodij ter ustrezno usklajenost med razvojno in testno ekipo. Zrele organizacije prehode programske rešitve med fazami DevOps preverjajo s kakovostnimi vrati (ang. Quality Gates) ter z zagotavljanjem neprekinjenega shranjevanja in uporabo metrik kvalitete. Sem spada čas za predelavo, število incidentov, število vrnitev na prejšnje stanje (ang. *Rollback*), čas iteracije razvojnega cikla in pokritost s testi.

- **Obvladovanje informatike** - doseženo raven obvladovanja infromatike ocenjujemo glede na definiranost procesov/procedur, njihovo standardiziranosti med različnimi projekti, nivo kontrolnih procesov, obstoj ekipe za upravljanje s procesi in uporabo metrik obvladovanja informatike. Kvalitetno obvladovanje informatike omogoča usklajeno delovanje vseh ostalih karakteristik in orientiranost oddelka proti doseganju glavnih poslovnih ciljev.

Elemente, definirane za potrebe ocenjevanje sprejetosti, smiselno razporedimo v faze. Tako prioritiziramo izboljšanje elementov, ki so del najmanj zrelih faz. Zrelost metodologije DevOps ocenjuje le osebje z ustreznim poznavanjem celotnega razvojno operativnega cikla organizacije. Do vprašalnika za ocenjevanje zrelosti naj bi tako imeli dostop le procesni inženirji. Vprašalnik je prikazan v tabeli 4.3.

3.3 Interpretacija rezultatov in oblikovanje izboljšav

Po izvedbi anketa bomo pridobljene ocene izbranih deležnikov razvojno operativnega cikla razvrstili v razsevni diagram, opisan in predlagan v delih [41, 42], ki smo ga prilagodili zahtevam našega modela. Abscisna in ordinatna os ne bosta predstavljali tehnične in sociološke ustreznosti, ampak neskladnost med mnenji skupin deležnikov. Na abscisi (os x) prikazujemo ocene sprejetosti elementov po mnenju procesnimi inženirji, na ordinati (os

y) pa sprejetost elementov med izvajalci. Ocenjeni elementi bodo predstavljeni kot točke na ustreznem mestu v diagramu. Vidik tehnične zrelosti faz cikla DevOps je prikazana v legendi z imeni, pripadajočimi barvami in vrednostjo, ki predstavlja povprečno oceno zrelosti te faze. Elementi na grafikonu bodo obarvani glede na to, kateri fazi pripadajo. Prilagojen razsevni diagram za razvrstitev elementov je prikazan na sliki 3.1.

Razsevni diagram je razdeljen na štiri dele (kvadrante) na podlagi povprečnih vrednosti sprejetosti med izvajalci in sprejetosti med procesnimi inženirji. Na podlagi teh kvadrantov razdelimo ocenjene elemente v štiri skupine:

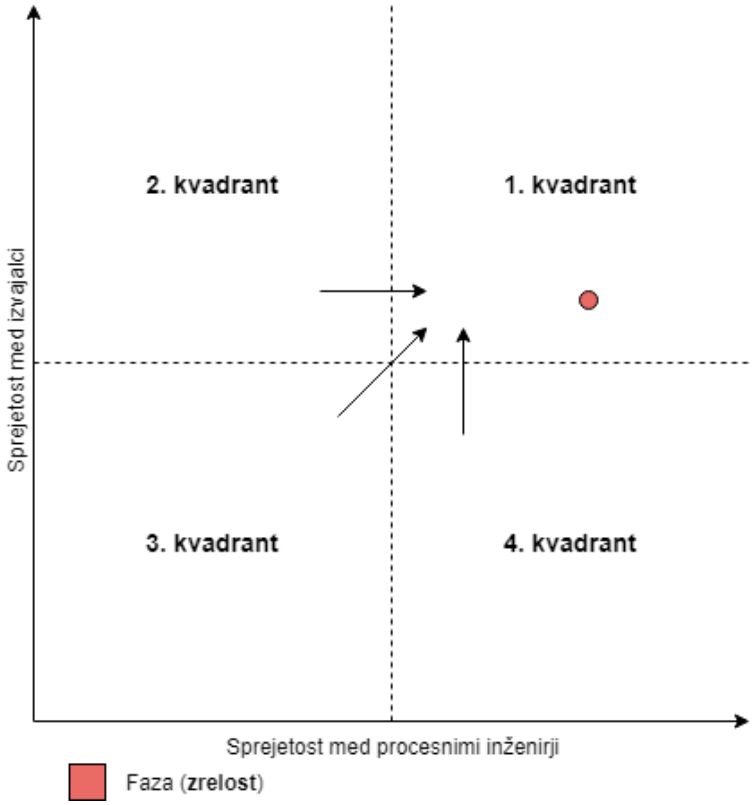
1. **Skupina prvega kvadranta** prikazuje elemente nadpovprečno sprejete tako s strani izvajalcev kot iz strani procesnih inženirjev. To pomeni, da je element povsem sociološko ustrezen in v proces razvoja programske opreme dobro integriran.

2. **Skupina drugega kvadranta** prikazuje elemente nadpovprečno sprejete med izvajalci in podpovprečno sprejete med procesnimi inženirji. Primer takega elementa je orodje, kateremu so izvajalci zelo naklonjeni, projektni vodje pa nimajo dostopa do ustreznih metrik uporabe orodja ter zato menijo, da ni dobro sprejeto.

3. **Skupina tretjega kvadranta** prikazuje elemente, ki so podpovprečno sprejeti s strani izvajalcev in procesnih inženirjev. To so sociološko neustrezni elementi. Te elemente je potrebno konkretno prilagoditi, v nekaterih primerih zamenjati ali celo odstraniti.

4. **Skupina četrtega kvadrant**a prikazuje elemente, ki so podpovprečno sprejeti s strani izvajalcev in nadpovprečno sprejeti s strani procesnih inženirjev. Primer takega elementa je orodje, v katerem procesni inženirji vidijo veliko pomembnost za celotni proces razvoja, a izvajalci za ustrezno uporabo niso dovolj usposobljeni.

Diagram omogoča hiter in podroben pregled razdelitve elementov v dvodimenzionalen prostor. Na podlagi zgoraj opisanih skupin ustvarimo vzorce



Slika 3.1: Razsevni diagram, ki prikazuje razporeditev elementov glede na sprejetost in zrelost. Barva elementa označuje fazo, v katero je element kategoriziran. Zrelost faze je s povprečjem ocen prikazana v legendi pod diagramom.

scenarijev za izboljšanje sprejetosti posameznega elementa. Scenarij predstavimo kot generičen skupek ukrepov za izboljšanje, ki pa ne upoštevajo konteksta ter dodatnih informacij (te dobimo iz pogovorov z uporabniki ali z rezultati ocenjevanja tehnične zrelosti faz procesa DevOps). Te vzorci služijo torej kot vstopna točka grajenja ukrepov za izboljšanje sprejetosti posameznih elementov. Vzorci so naslednji:

- V primeru elementa drugega kvadranta želimo povečati sprejetost elementa med procesnimi inženirji in ohraniti sprejetost elementa med izvajalci. Glede na ocene posameznih karakteristik je možni ukrep

povečanje števila priložnosti za uporabo ali izboljšanje povezanosti oziroma integracije elementa z drugimi orodji in aktivnostmi. Nadgrajevanje infrastrukture, ki element vzdržuje, lahko pozitivno vpliva na metrike uspešnosti elementa. Te so za procesne inženirje, ki po možnosti aktivnosti dnevno ne izvajajo, ključen uvid v delovanje elementa.

- Elementi tretjega kvadranta so potrebni konkretne prenove. Izvajalci in procesni inženirji so usklajenega mnenja, da je element sociološko neustrezen. V primeru, da element ni nepogrešljiv, je prva možnost, da element odstranimo iz metodologije DevOps. V drugih primerih pa želimo element konkretno prenoviti. Zamenjamo lahko orodje ali pa korenito prenovimo aktivnost.

- V primeru elementa četrtega kvadranta želimo povečati sprejetost elementa med izvajalci in ohraniti dobro sprejetost med procesnimi inženirji. To se lahko doseže z zvišanjem uporabe elementa ob dani priložnosti. Če ugotovimo, da je glavni razlog za nesprejetost med izvajalci pomanjkanje znanja, se posvetimo izobraževanju izvajalcev o pravilni rabi in prednostih tega elementa. Izboljšamo dokumentacijo in bolj podrobno definiramo aktivnost oziroma proces, povezan z elementom. Nadalje želimo izvajalcem zagotoviti dostop do orodij za spremljanje metrik tega elementa in do komunikacije s sodelavci, za katere je uporaba tega elementa pomembna v kasnejših delih razvojnega cikla.

Za boljši kontekst in dodatne informacije pri gradnji scenarijev izboljšav uporabimo vidik ocenjevanja zrelosti faz DevOps. Razvojno operativni cikel deluje le tako hitro kot deluje njegova najpočasnješa faza. Eden glavnih principov cikla DevOps je identifikacija in povzdigovanje omejitev [26]. V našem model omejitve identificiramo s pomočjo analize ocenjevanja tehnične zrelosti posameznih faz razvojno operativnega cikla DevOps. Elemente faz, ki se izkažejo za najmanj zrele, je potrebno obravnavati prioritetno, saj imajo največ negativnega vpliva na hitrost in kakovost delovanja celotnega cikla. Za nezrele faze je potrebno tudi premisliti o vpeljavi novih elementov.

Po apliciranju kontekstualiziranih ukrepov za izboljšanje želimo, da se bi elementi ob ponovni izvedbi raziskave sprejetosti pomaknili proti prvemu kvadrantu razsevnega diagrama. Torej bi se splošna povprečna sprejetost elementov znotraj organizacije povečala.

3.4 Pristop k izvedbi ocenjevanja

V tem podpoglavju so predstavljeni koraki za izvedbo našega modela.

V prvem koraku se s podjetjem spoznamo in dobimo vpogled v proces DevOps. Identificiramo orodja in aktivnosti, ki jih organizacija uporablja in njihovo razumevanje razdelitve metodologije DevOps na faze. Na podlagi obstoječe dokumentacija ter s pomočjo pogovora s skrbnikom metodologije DevOps ustvarimo diagram elementov in faz njihovega razvojno operativnega cikla.

V drugem koraku se pripravi vprašalnike po vzorcih, prikazanih v tabelah 3.1, 3.2 in 3.3. Izvajalci odgovarjajo na vprašanja, navedena v tabeli 3.1, procesni inženirji pa na vprašanja v tabelah 3.2 in 3.3. Vprašalnike se ustvari tako, da se njihova vsebina dinamično prilagaja vlogi uporabnika. Trditve najdene v vprašalnikih uporabniki vrednotijo z Likertovo sedemstopenjsko lestvico (ang. Likert scale), ki je razširjena lestvica za vrednotenje te vrste vprašalnikov [9]. Za vrednotenje karakteristike pogostost uporabe ob priložnosti uporabimo nekoliko prirejeno lestvico.

V tretjem koraku s pomočjo vodje IT oddelka identificiramo vloge in odgovornosti uporabnikov metodologije. Vloge razdelimo v dve skupini, na katerih temelji razdelitev vprašalnikov. Med izvajalce uvrstimo razvijalce, člane ekipe za nadzor kakovosti, oblikovalce, sistemske inženirje itn. Med procesne inženirje pa spadajo vodje ekip, vodja IT oddelka ter, če je ta vloga v podjetju definirana, tudi dejanski procesni inženirji. V tem koraku točno določimo, katere vloge posamezne elemente uporabljajo. Anketo se izvede tako, da na vprašanja o posameznem elementu odgovarjajo le tisti uporabniki, ki ta element dejansko dobro poznajo.

Ocenjevanje sprejetosti med izvajalci	
Pogostost uporabe	Kako pogosto ob pojavitvi priložnosti za izvedbo <<element>> to tudi izvedete?
Relativna prednost	Izvajanje <<element>> izboljša mojo produktivnost. Izvajanje <<element>> izboljša kvaliteto mojega dela.
Kompatibilnost	Izvajanje <<element>> je kompatibilno z mojim načinom dela.
Kompleksnost	Izvajanje <<element>> je enostavno in dobro dokumentirano.

Tabela 3.1: Vprašalnik za ocenjevanje sprejetosti elementov metodologije DevOps za izvajalce.

Sledi analiza rezultatov vprašalnikov. Sprejetost elementov predstavimo v razsevnem diagramu ter za izbrane elemente in faze pripravimo ukrepe za izboljšanje sprejetosti oziroma zrelosti. Na koncu rezultate in analizo predstavimo skrbniku metodologije DevOps v organizaciji.

Ocenjevanje sprejetosti ekipe s strani procesnih inženirjev	
Relativna prednost	Izvajanje <<element>> izboljša produktivnost ekipe. Izvajanje <<element>> izboljša kvaliteto dela ekipe.
Kompatibilnost	Izvajanje <<element>> je kompatibilno z načinom dela ekipe.
Kompleksnost	Izvajanje <<element>> je enostavno in dobro dokumentirano.

Tabela 3.2: Vprašalnik za ocenjevanje sprejetosti elementov metodologije DevOps za procesne inženirje.

Ocenjevanje zrelosti	
Avtomatizacija	Stopnja avtomatizacije <<faza>> je v primerjavi s sorodnimi podjetji/IT oddelki zelo visoka.
Sodelovanje	Stopnja sodelovanja in deljenja znanj v <<faza>> je v primerjavi s sorodnimi podjetji/IT oddelki zelo visoka.
Kvaliteta	Stopnja kvalitete dela in izdelkov v <<faza>> je v primerjavi s sorodnimi podjetji/IT oddelki zelo visoka.
Obvladovanje informatike	Stopnja obvladovanja informatike in postopanja po formalnih procesih v <<faza>> je v primerjavi s sorodnimi podjetji/IT oddelki zelo visoka.

Tabela 3.3: Vprašalnik za ocenjevanje zrelosti faz metodologije DevOps.

Poglavje 4

Študija primera

Študija primera je bila izvedena za potrditev koristnosti izdelanega modela predstavljenega v poglavju 3. V tem poglavju bomo opisali organizacijo, ki nam je izvedbo študije omogočila, predstavili identificirane elemente in faze metodologije DevOps ter opisali potek izvedbe študije. Prikazali bomo rezultate analize in predlagali možne ukrepe za izboljšanje stanja določenih elementov ali faz. V zadnjem podpoglavju bomo predstavili mnenja zaposlenih.

4.1 Opis podjetja

Študija primera je bila izvedena v večjem slovenskem podjetju z obširnim IT oddelkom. Oddelek razvija in vzdržuje spletne aplikacije, spletne strani, informacijski sistem za upravljanje vsebin in druge programske rešitve. Ločena ekipa, ki hierarhično spada sicer med razvojne ekipe, skrbi za uvedbo in vzdrževanje platforme DevOps. Glede zrelosti njihove metodologije DevOps menijo, da so v še relativno zgodnjih fazah. Pristop DevOps se je v podjetje začelo uvajati pred dobrima dvema letoma in razvijanje platforme je danes še v polnem teku.

DevOps ekipa trenutno primarno skrbi za platformo neprekinjene integracije in dostave, ki mora zaradi velikosti oddelka in razpona projektnih

tipov podpirati številne programske jezike in ogrodja. Vsak tip projekta ima nekoliko prilagojene postopke za grajenje in testiranje. Temelj platforme predstavlja orodje za neprekinjeno integracijo Jenkins. Ta zagotovi izvedbo cevovodov, ki so si različni med tipi programskih rešitev. Za upravljanje verzij in izdaj uporabljajo IBMov UrbanCode. Uporabljajo prakso neprekinjene dostave. Programske rešitve se v testna okolja izdajajo preko CI cevovoda avtomatsko, premik v produkcijsko okolje pa še vedno zahteva ročno posredovanje. Glede spremljanja cikla se v veliki meri naslanjajo na orodja Elastic Stack. Za planiranje, prioritiziranje, beleženje aktivnosti, deljenje znanja in dokumentacijo pa uporabljajo kombinacijo Atlassianovih orodij Jira in Confluence.

Vodja te ekipe je skrbnik platforme DevOps in z njegovo pomočjo smo ustvarili razdelitev platforme na elemente in faze. Vodja IT oddelka pa je poskrbel za razdelitev anket zaposlenim s primernimi vlogami in odgovornostmi, tako da smo dobili odgovore za vse elemente metodologije. Končna predstavitev in razgovor je bila opravljena z vodjo DevOps ekipe.

4.2 Izvedba študije

Študijo smo izvedli s pristopom opisanim v podpoglavju 3.4. V prvem koraku smo preko sestankov želeli identificirati ustrezno razdelitev njihove metodologije DevOps. Po uvodnem sestanku smo dobili osnoven pogled v delovanje in dokumentacijo njihovega cikla DevOps. Izdelali smo osnutek seznama faz in elementov. Drugi sestanek je služil za pregled in potrditev seznama. Nekatere elemente smo priredili za boljše razumevanje, nekatere pa iz seznama odstranili. Primera slednjih elementov sta bila statična analiza kode, ki se je ob izvedbi modela še uvajala, in pa orodje Tasktop, ki v trenutku izpeljave modela, kljub ustrezni integraciji, še ni imel dovolj uporabnikov. Seznam smo še dopolnili z nekaterimi elementi za spremljanje, ki smo jih pri analizi dokumentacije spregledali. Seznam elementov se je skrčil iz začetnih triindvajsetih na končnih dvajset elementov. Celoten seznam elementov je

prikazan v tabeli 4.1. Metodologije pa smo razdelili še na 5 faz:

1. Ustvarjanje - zajema načrtovanje in oblikovanje rešitev, kodiranje in vejenje, upravljanje z izvorno kodo, grajenje programskih rešitev, funkcionalne teste in izdelavo kandidatov za izdajo.

2. Predprodukcija - sem spada CI, orodja za avtomatizirano testiranje ter dostava kode in programskih rešitev.

3. Izdaja - zajema neprekinjeno dostavo oziroma postavitev, urnik izdaj, uvedbo aplikacij, status uvedbe, kontrole za obnovitev aplikacij/sistemov in podobno.

4. Spremljanje - zajema zbiranje povratnih informacij, nadzorovanje obremenitev, zbiranje informacij končnih uporabnikov ter metrike uspešnosti in razpoložljivosti.

5. Planiranje - planiranje, uporaba zbranih povratnih informacij, poslovne metrike, prioritiziranje funkcionalnosti in popravkov, nadzor dela v teku in podobno.

Po potrditvi obeh seznamov s strani podjetja smo pripravili vprašalnike. Za potrebe te študije smo ustvarili univerzalni vprašalnik, ki je prikaz vprašanj dinamično prirejal glede na izbrano vlogo v prvem vprašanju in poznavanju posameznega elementa. Za ustvarjanje in izvedbo smo uporabili spletno aplikacijo 1KA [1], ki omogoča tudi osnovno analizo rezultatov in za nas ustrezne možnosti izvoza podatkov. Vprašanja, predstavljena v tabelah 3.1, 3.2 in 3.3 smo zaradi jasnosti in slovnične doslednosti za nekatere elemente morali nekoliko prilagoditi. Po posvetu z organizacijo smo k vprašalnikom dodali vprašanje odprtega tipa, kjer bi razvijalci lahko podali svoje predloge za izboljšanje aktivnosti ali zamenjavo orodij.

Po potrditvi vprašalnika s strani organizacije smo v sodelovanju z vodjo IT oddelka ustvarili seznam uporabnikov, katerim bomo anketo poslali. Anketo smo aktivirali, udeleženci pa so povabilo za sodelovanje dobili preko

E1	Vejenje in verzioniranje z GitFlow-om in semantičnim verzioniranjem
E2	Razvijanje v prednastavljeni virtualki
E3	Proženje cevovoda preko [ci] ukazov
E4	Razvoj z Idea IntelliJ orodjem
E5	Java razvoj z Eclipse orodje
E6	Razvoj z Visual Studio Code-om
E7	Implementacija cevovodov neprekinjene integracije (CI) z orodjem Jenkins
E8	Arhitektura mikrostoritev z Docker kontejnerizacijo
E9	Testiranje Java enot z orodjem Junit
E10	Integracijsko testiranje z orodjem Arquillian
E11	Konfiguracija okolij in uvajanje aplikacij z orodjem UrbanCode Deploy
E12	Konfiguracija okolij in uvajanje aplikacij z orodjem Ansible
E13	Opozarjanje in spremljanje zdravja aplikacij z orodjem Elastic Stack
E14	Spremljanje zdravja Docker kontejnerjev z orodjem Portainer
E15	Spremljanje zdravja omrežij in strežnikov z orodjem Nagios XI
E16	Spremljanje zdravja omrežij in strežnikov z orodjem Icinga
E17	Spremljanje varnostnih napadov in splošne varnosti z orodjem QRadar
E18	Sledenje zahtevkom in planiranje dela z orodje Jira
E19	Medekipno komuniciranje z orodjem Slack
E20	Delitev znanja in načrtov z orodjem Confluence

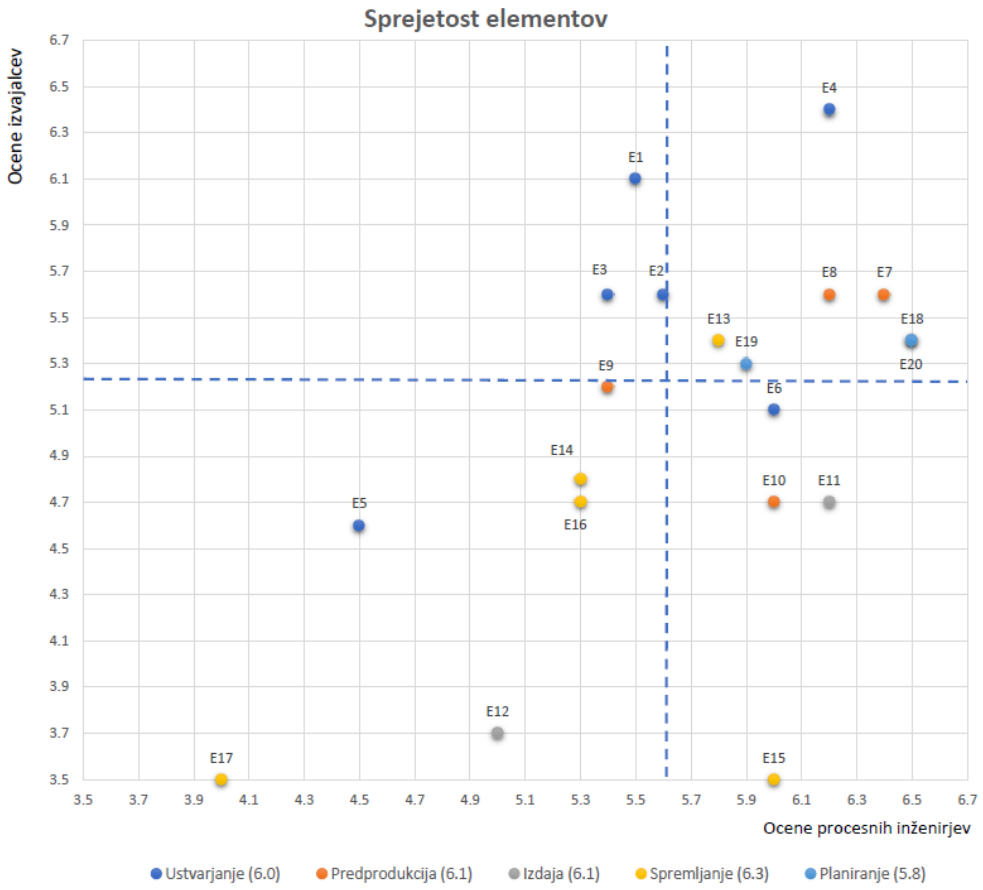
Tabela 4.1: Seznam elementov podjetja.

elektronske pošte. Odzive smo zbirali tri tedne. Dobljene podatke smo za analizo izvozili v Microsoft Excel [11], ki nam je omogočil izdelavo grafikonov in tabel. Rezultati analize podatkov ter diskusija rezultatov so podrobno predstavljeni v naslednjih poglavjih.

4.3 Rezultati analize podatkov

Izbrani uporabniki so elemente in faze ocenili s pomočjo modela opisanega v poglavju 3. Njihove ocene so zbrane v tabelah 4.2 in 4.3, razporeditev elementov pa je vidna na diagramu na sliki 4.1. Pri anketiranju je sodelovalo 31 uporabnikov. Od tega je bilo 8 procesnih inženirjev in 23 izvajalcev. Med uporabniki so prevladovali razvijalci oziroma vodje ekip razvijalcev. Teh je bilo 21. V anketo so bili vključeni še trije člani ekipe za zagotavljanje kakovosti, trije člani ekipe za podporo delovanju in trije sistemski inženirji.

Število odgovorov za posamezni element se razlikuje glede na število in



Slika 4.1: Razsevni diagram, ki prikazuje razporeditev elementov po izvedenem ocenjevanju. Seznam elementov skupaj s polnimi imeni se nahaja v tabeli 4.1. Barva posameznega elementa znotraj diagrama simbolizira fazo, ki ji element pripada. Povprečne ocene zrelosti faz so prikazane v legendi v oklepajih.

	Sprejetost med izvajalci						Sprejetost med procesnimi inženirji				
E	I1	I2	I3	I4	I5	Povp.	PI1	PI2	PI3	PI4	Povp.
E1	7.0	5.8	5.8	6.2	5.8	6.1	5.5	6.3	5.8	4.5	5.5
E2	6.0	6.0	5.5	4.5	6.0	5.6	6.0	5.5	5.5	5.5	5.6
E3	5.1	5.6	5.8	5.6	5.5	5.6	5.7	5.7	5.3	4.7	5.4
E4	6.7	6.5	6.5	5.9	6.2	6.4	6.3	6.3	6.3	6.0	6.2
E5	4.3	4.7	4.7	4.7	4.7	4.6	4.5	4.5	5.0	4.0	4.5
E6	4.4	5.1	5.1	5.2	5.7	5.1	6.0	6.0	6.0	6.0	6.0
E7	5.4	5.8	5.8	5.4	5.3	5.6	6.3	6.3	6.7	6.3	6.4
E8	5.8	5.6	5.7	5.4	5.5	5.6	6.0	6.3	6.3	6.0	6.2
E9	4.3	5.2	5.8	5.2	5.3	5.2	5.0	6.5	5.0	5.0	5.4
E10	3.5	4.8	5.8	4.5	4.8	4.7	6.0	6.0	6.0	6.0	6.0
E11	5.3	5.0	4.3	5.7	3.3	4.7	6.0	6.3	6.3	6.3	6.2
E12	2.5	4.0	4.0	4.0	4.0	3.7	4.5	5.5	5.0	5.0	5.0
E13	4.6	5.5	5.6	5.6	5.5	5.4	5.7	6.0	5.7	5.7	5.8
E14	5.0	4.5	5.0	5.0	4.5	4.8	5.3	5.3	5.3	5.3	5.3
E15	///	///	///	///	///	///	6.0	6.0	6.0	6.0	6.0
E16	3.7	4.3	4.3	5.7	4.3	4.7	5.0	5.5	5.5	5.0	5.3
E17	///	///	///	///	///	///	4.0	4.0	4.0	4.0	4.0
E18	6.1	5.3	5.0	5.3	5.5	5.4	5.7	6.7	6.7	6.7	6.5
E19	6.0	5.3	4.7	5.4	5.0	5.3	6.3	5.3	6.0	6.0	5.9
E20	5.3	5.3	5.5	5.3	5.3	5.4	6.0	6.7	6.7	6.7	6.5
Povp.						5.2					5.6

Tabela 4.2: Rezultati ocenjevanja sprejetosti elementov. Stolpec E prikazuje elemente metodologije podjetja, stolpci I oziroma PI pa povprečne vrednosti odgovorov na posamezna vprašanja. Pet stolpcev (I1-I5) prikazuje odgovore izvajalcev, štiri pa odgovore procesnih inženirjev (PI1-PI4). Obe skupini vprašanj sta zaključeni s povprečno vrednostjo vseh vprašanj.

Faza	V1	V2	V3	V4	Povp.
Ustvarjanje	6.3	5.7	5.7	6.3	6.0
Predprodukcija	6.3	5.7	5.7	6.7	6.1
Izdaja	6.3	5.7	6.0	6.3	6.1
Nadzorovanje	6.0	6.7	6.3	6.3	6.3
Planiranje	5.7	5.7	5.7	6.0	5.8

Tabela 4.3: Rezultati ocenjevanja zrelosti faz. Stolpci V prikazujejo povprečno vrednost odgovorov za posamezno karakteristiko pri ocenjevanju zrelosti.

dosegljivost uporabnikov, ki so ta element poznali in ga aktivno uporabljali. Največ odgovorov za posamezni element smo zbrali za element E18, najmanj za nekatera orodja spremljanja. Primera takih elementov sta E15 in E17, ki ju v podjetju zaenkrat uporabljajo izključno sistemski inženirji. V diagramu sta ta elementa prikazana na osi x le glede na ocene procesnih inženirjev, saj mnenj izvajalcev nismo uspeli pridobiti. To je posledica naše napačne predpostavke, da je uporaba elementov razširjena preko celotnega IT oddelka. Ekipa sistemskih inženirjev zaenkrat še ni popolnoma vključena v metodologijo DevOps, pri sestavljanju seznama pa smo dajali predvsem poudarek na uporabnikih, ki poznajo cikel DevOps v celoti, tako da smo nekaj sistemskih inženirjev izpustili.

Uporabniki so elemente večinoma ocenjevali pozitivno. Povprečna ocena elementa znaša 5.4. Izvajalci so bili glede ocenjevanja sprejetosti zelo usklajeni s procesnimi inženirji. To lepo prikaže diagram na sliki 4.1. Odstopanja med ocenami enih in drugih ni veliko. Elementi so po diagramu razporejeni zelo linearno z redkimi izjemami. Vseeno so bile povprečne ocene sprejetosti s strani izvajalcev kanček nižje kot s strani procesnih inženirjev.

Na vidik zrelosti faz metodologije DevOps v podjetju smo zbrali odgovore treh procesnih inženirjev. Najboljše je bila ocenjena faza spremljanja, najslabše pa faza planiranja. Pri tem je zanimivo, da je bila sprejetost elementov, ki so del faze planiranja, povprečno visoka. Elementi faze spremljanja pa so bili glede sprejetosti ocenjeni najslabše.

Na podlagi pozicij elementov v diagramu in zrelosti faz smo izbrali elemente, ki imajo po našem mnenju največ potenciala za izboljšavo. Primarno smo želeli izboljšati sprejetost elementov drugega, tretja in četrtega kvadranta diagrama. Osredotočili smo se tudi na elemente faze planiranja, saj je bila ta glede vidika zrelosti najslabše ocenjena. S predlaganimi ukrepi smo želeli izboljšati sprejetost elementa pri skupini uporabnikov, kjer je bil slabo sprejet, ali pa izboljšati splošno zrelost faze kateri pripada.

4.3.1 (E10) Integracijsko testiranje z orodjem Arquillian

Element se nahaja v četrtem kvadrantu, torej je bil podpovprečno ocenjen s strani izvajalcev in nadpovprečno s strani procesnih inženirjev. Integracijsko orodje smo med pogovori za identifikacijo uvrstili v fazo predprodukcije, kar načeloma pomeni, da je integracijsko testiranje vključeno v cevovod za neprekinjeno integracijo. Analiza odgovorov in kasnejša diskusija rezultatov je razkrila, da temu zaenkrat še ni tako. Predvsem se je izkazalo, da nekateri izvajalci ob priložnosti integracijsko testiranje izvajajo redko. Ob pogovoru z vodjo platforme DevOps smo ugotovili, da ga razvijalci še vedno izvajajo ročno. Logičen ukrep je torej vključitev te vrste testiranja v CI cevovod. To potrjuje tudi primerjava z elementom E9 (testiranje Java enot z orodjem JUnit). To testiranje je namreč že delno vključeno v CI cevovod in tudi nekoliko boljše sprejeto s strani izvajalcev. Rezultati ocenjevanja sprejetosti elementa E10 razkrijejo še podpovprečne ocene izvajalcev glede karakteristike kompleksnost. To bi lahko izboljšali z dodatnim izobraževanjem izvajalcev o uporabi in pomembnosti integracijskega testiranja.

4.3.2 (E11) Konfiguracija okolij in uvajanje aplikacij z orodjem UrbanCode Deploy

Z IBMovim orodjem UrbanCode Deploy podjetje skrbi za izdajo programske opreme. Uporaba orodja je odvisna od vloge posameznika. Za potrebe modela smo sprejetost orodja ocenjevali primarno iz vidika članov DevOps ekipe, ki poleg same izdaje aplikacij skrbijo še za ustrezno konfiguracijo in vzdrževanje orodja, komponent in okolij. Element se nahaja v četrtem kvadrantu, kar pomeni, da je podpovprečno ocenjen s strani izvajalcev in nadpovprečno s strani procesnih inženirjev. Preko pogovorov s skrbnikom platforme DevOps smo izvedeli, da je eden izmed ključnih orodij celotne platforme in je nezamenljiv. Na orodju namreč temelji praktično celotna izdaja programske opreme. Izvajalci so ga ocenili kot slabega predvsem glede

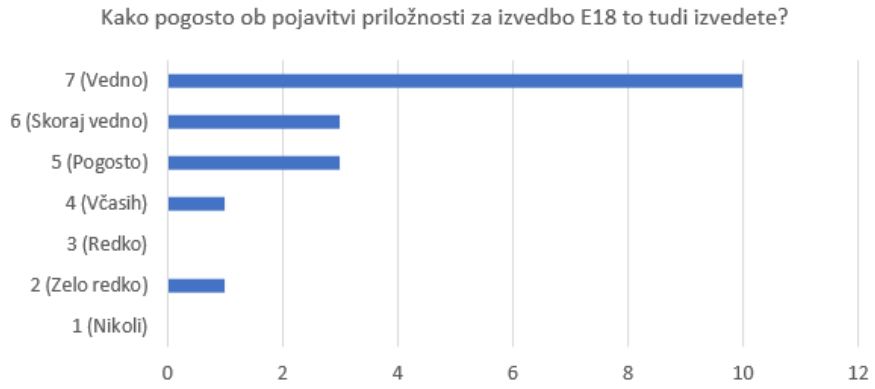
karakteristike kompleksnost, kar pomeni, da menijo da orodje ni enostavno in dobro dokumentirano. Glede na obširnost nalog orodja in izdatno število uporabljenih vtičnikov, se nam ta ocena zdi upravičena. V tem delu priporočamo dodatno izobraževanje izvajalcev. Izvajalci so tudi mnenja, da element ne vpliva pozitivno na kvaliteto dela. Po analizi načina dela z orodjem smo ocenili, da bi bil primeren ukrep izboljšanje dokumentacije procesa dela. S tem bi omogočili konstantno sledenje najboljšim praksam uporabe orodja in večjo konstantnost pri njegovi uporabi.

4.3.3 (E14) Spremljanje zdravja Docker vsebnikov z orodjem Portainer

Portainer je orodje z grafičnim vmesnikom za upravljanje in spremljanje Docker vsebnikov. V podjetju ga uporabljajo večinoma razvijalci in DevOps inženirji za lažji pregled nad Docker vsebniki in njihovo konfiguracijo. Element je bil ocenjen podpovprečno tako s strani izvajalcev kot s strani procesnih inženirjev. Funkcionalnosti Portainerja je možno v določeni meri nadomestiti s preprostimi orodji ukazne vrstice, zato ni ključnega pomena. Preko pogovorov nam je procesni inženir omenil tudi, da so priložnosti za uporabo orodja relativno redke. Izvajalci so najslabše ocenili vpliv orodja na njihovo produktivnost in ga opišejo kot nekoliko nestabilnega. Logičen ukrep glede tega je izboljšanje stabilnosti orodja. Ob upoštevanju, da orodje ni ključnega pomena, predlagamo tudi razmislek o zamenjavi orodja s kakšno izmed alternativ. Ob potencialni vpeljavi sistema Kubernetes pa predlagamo odstranitev Portainerja v celoti.

4.3.4 (E16) Spremljanje zdravja omrežij in strežnikov z orodjem Icinga

Icinga je odprtokodno orodje za spremljanje računalniških sistemov in omrežij. Za ta namen ga skupaj z orodjem Nagios XI primarno uporabljajo sistemski inženirji. Podpovprečno je bil ocenjen v obeh skupinah osebja in spada



Slika 4.2: Prikaz porazdelitve odgovorov na vprašanje karakteristike pogostost uporabe ob priložnosti za element E18.

v tretji kvadrant razsevnega diagrama. Je eno od slabo sprejetih orodij za spremljanje, kljub dobro ocenjeni zrelosti same faze spremljanja. Najbolje je bil ocenjen po karakteristiki kompatibilnost, kar pomeni da uporabniki dojemajo element kot skladnega z njihovimi navadami in izkušnjami. Aktivnost bi bilo torej logično obdržati. Predlagamo pa izbiro in ohranitev le enega izmed orodij Nagios XI in Icinga, saj se njune funkcionalnosti v večini prekrivajo. Zaradi majhnega števila ocen obeh elementov bi bila za ustrezno izbiro potrebna dodatna analiza uporabe teh dveh orodij.

4.3.5 (E18) Sledenje zahtevkom in planiranje dela z orodjem Jira ter faza planiranja

Element E18 se sicer nahaja v prvem kvadrantu, torej je dobro sprejet s strani izvajalcev in s strani procesnih inženirjev. Za podrobnejšo analizo smo se odločili zaradi slabe ocene zrelosti faze planiranja in velikega števila uporabnikov. V primerih elementov, ki jih uporablja praktično celotni IT oddelek podjetja, je potencial tudi manjših izboljšav velik. Jira je priljubljeno orodje, v osnovi narejeno za sledenje zahtevkom in projektno vodenje. Z

Slika 4.3: Prikaz porazdelitve odgovorov izvajalcev na vprašanja karakteristik relativna prednost (zgoraj levo in zgoraj desno), kompatibilnost (spodaj levo) in kompleksnost (spodaj desno) za element E18.

dodajanjem vtičnikov lahko pokrije večino aktivnosti faze planiranja, kot so upravljanje osebja in urnikov, napredna prioritizacija ter integracija z orodji za spremljanje programske opreme. Sprejetost orodja je visoka predvsem med procesnimi inženirji, med izvajalci pa je mnenje nekoliko bolj deljeno. Porazdelitve odgovorov izvajalcev prikazujejo slike 4.2 in 4.3. Opazimo lahko oblikovanje dveh skupin. Prva številčnejša skupina uporabo Jire oceni nadpovprečno in v skladu z ocenami procesnih inženirjev, skupina treh uporabnikov pa aktivnost konsistentno ocenjuje z nizkimi ocenami. Predvsem so usklajeni glede vprašanj karakteristike relativna prednost. Menijo, da Jira vpliva negativno tako na kvaliteto njihovega dela kot na njihovo produktivnost. Tu predlagamo deljenje znanja in načinov uporabe med skupinama. Skupina, ki orodje ocenjujejo kot slabo, bi tako dobila vpogled v način ustrezne rabe orodje. Za začetek pa bi bil potreben dodaten sestanek s to manjšo skupino uporabnikov za razjasnitev njihovih pomislekov. Zaradi pomembnosti orodja za fazo planiranja predlagamo tudi splošno podrobnejšo analizo uporabe orodja in razširjanje obsega uporabe.

4.4 Predstavitev rezultatov vodji

Ob predstavitvi rezultatov izvedbe modela smo preko vprašanj zbirali mnenje vodje DevOps ekipe o sestavnih delih modela in o modelu na splošno. Predstavitev in diskusija sta bili izvedeni po analizi zbranih podatkov. V tem podpoglavju bomo predstavili odziv na ocenjevanje sprejetosti, ocenjevanje zrelosti in na predlagane ukrepe za izboljšanje trenutnega stanja.

Ob prikazu razsevnega diagrama z razporeditvijo elementov ga je najbolj presenetil element E10. Strinja se s potrebo po vključitvi integracijskega testiranja v CI cevovod. Pravi, da je trenutno platforma za neprekinjeno testiranje še v izdelavi. Ta bo vsebovala več vrst avtomatičnega testiranja (tudi na primer integracijsko testiranje in statično analizo kode). Strinja se tudi s tem, da je potrebno razvijalce o tej vrsti testiranja dodatno izobraziti, a bodo taki ukrepi aktualni šele po postavitvi platforme. Glede ocenjevanja elementa E11 je izpostavil manjšo napako pri izdelavi seznama elementov. UrbanCode Deploy smo namreč dali v ocenjevanje le DevOps inženirjem, v nekoliko drugačnem obsegu pa ga uporabljajo tudi razvijalci. Smiselno bi bilo ocenjevanje dveh elementov, ki bi pokrila uporabo pri obeh ekipah. Ne preseneča ga dejstvo, da se orodje zdi upravljavcem kompleksno. Strinja pa se predvsem z ukrepom izboljšanja dokumentacije procesa dela. Meni, da so sicer DevOps inženirji o uporabi orodja dovolj izobraženi. Glede možnih ukrepov za izboljšanje sprejetosti uporabe orodja Portainer (E14) pravi, da drastične spremembe tega elementa naj trenutno ne bi bile prioriteta. Strinja pa se s potrebo po večji stabilnosti orodja. Kot razlog za uporabo orodij Icinge in Nagios XI (E16 in E15) pa navaja nikoli do konca izveden prehod med orodjema. Meni, da bi bila za identifikacijo potrebnih ukrepov potrebna dodatna analiza uporabe izključno med sistemskimi inženirji.

Ocenjevanje zrelosti faz je potekalo po pričakovanjih vodje DevOps ekipe. Glede podpovprečnih ocen za fazo planiranja pravi, da kot oddelek zaenkrat še niso sposobni v celoti uporabljati orodja Jira. Zaenkrat ga uporabljajo le za razdelitev zahtevkov, ne pa za planiranje oziroma upravljanje z osebjem in drugimi viri. Meni, da je orodje Jira sicer primerno in več kot sposobno po-

krivati vse zahteve faze. Za nadpovprečno ocenjevanje faze spremljanja meni, da je razlog za to najverjetneje dobra integriranost Elastic Stacka (element E13).

Splošno je podal o modelu pozitivno mnenje. Meni, da je dobro povzel realno sliko trenutnega stanja. Najbolj ga je presenetila nizka stopnja izvajanja integracijskega testiranja z orodjem Arquillian in pa slaba ocenjenost elementa E3 s strani procesnih inženirjev. Načeloma se mu zdijo rezultati in razdelitev elementov smiselni, hkrati pa doda, da mu je model omogočil boljši vpogled v razloge za to. Sprejel je tudi šest izmed predlaganih ukrepov. Od tega sta bila dva planirana že pred uporabo mojega modela, dva pa je dodal v kratkoročni plan ekipe DevOps po uporabi modela. Integracijsko testiranje in orodje Arquillian (element E10) bo sedaj večji del planirane platforme za neprekinjeno testiranje. Sprejel je še ukrepe za elementa E11 in E18, a pravi da jih uvršča v dolgoročne plane. Uporabo orodja UrbanCode Deploy (element E11), naj bi bila zaenkrat še na zadovoljivi ravni, za uvajanje sprememb v proces uporabe Jire pa bo potrebna podrobna specializirana raziskava.

Poglavje 5

Zaključek

V diplomskem delu smo izdelali in opisali model za ocenjevanje metodologije v podjetjih, ki uporabljajo DevOps. Najprej smo poglavju 2 raziskali literaturo ključnih področij naše diplomske naloge. Pregledali smo glavne principe DevOps in predstavili faze, prakse in aktivnosti, ki metodologijo DevOps navadno sestavljajo. Raziskali smo še področje ocenjevanja metodologij razvoja programske opreme. Pregledano literaturo smo nato uporabili za izgradnjo našega modela v poglavju 3. V poglavju 4 smo opisali, kako je potekal preizkus modela s študijo primera v slovenskem podjetju.

Model s pomočjo ocenjevanja sprejetosti posameznih elementov in zrelosti faz omogoča vpogled v delovanje celotnega razvojno operativnega cikla in identifikacijo elementov, ki so najbolj potrebni izboljšav. Z ločitvijo uporabnikov v dve skupini smo ustrezne izboljšave lažje prepoznali. Izvedba študije primera nam je omogočila testiranje delovanja modela v praksi. Podjetje, ki je v študiji sodelovalo, je potrdilo, da model prikazuje realno sliko metodologije in omogoča boljši vpogled v razloge za trenutno stanje. Prav tako so sprejeli šest izmed naših predlaganih ukrepov za izboljšanje. Dva ukrepa so celo uvrstili v kratkoročni plan. Tudi ločeno ocenjevanje metodologije DevOps glede na faze in elemente se je izkazalo za smiselno. Omogočilo je identifikacijo elementov, ki so potrebni izboljšanja kljub na prvi vtis dobri sprejetosti, ter ustrezno prioritizacijo predlaganih ukrepov.

Izvedena študija primera nam je pomagala odkriti tudi nekatere omejitve modela. Model metodologijo teoretično razdeli na faze, ki vsebujejo posamezne elemente, ki so lahko orodja ali aktivnost. A v praksi se izkaže, da veliko aktivnosti oziroma orodij presega okvire ene faze. To v nekaterih primerih naredi primerjavo ocen zrelosti neke faze s povprečnimi ocenami elementov te faze nesmiselno. Za dodatno zmedo lahko poskrbi različno dojemanje razdelitve med zaposlenimi. Kljub sodelovanju z osebjem podjetja pri razdelitvi vsak član posamezne faze vseeno dojema nekoliko samosvoje. Splošno pa je največja pomanjkljivost diplomske naloge le ena izvedena študija primera. Za temeljito testiranje modela bi bilo potrebno izvesti še nekaj drugih v organizacijah različnih vrst, velikosti in z različnimi stopnjami zrelosti metodologije DevOps.

Literatura

[1] 1ka. Dosegljivo: https://www.1ka.si. Dostopano: 09.09.2019.

[2] Andon (manufacturing). Dosegljivo: https://en.wikipedia.org/wiki/Andon_(manufacturing). Dostopano: 30.06.2019.

[3] Bamboo. Dosegljivo: https://www.atlassian.com/software/bamboo. Dostopano: 08.09.2019.

[4] Confluence. Dosegljivo: https://www.atlassian.com/software/confluence. Dostopano: 08.09.2019.

[5] Devops roadmap. Dosegljivo: https://roadmap.sh/devops. Dostopano: 08.09.2019.

[6] Devops topologies. Dosegljivo: https://jenkins.io/. Dostopano: 08.09.2019.

[7] Jenkins. Dosegljivo: https://aws.amazon.com/devops/what-is-devops/. Dostopano: 30.06.2019.

[8] Jenkins plugins. Dosegljivo: https://plugins.jenkins.io. Dostopano: 30.06.2019.

[9] Likert scale. Dosegljivo: https://en.wikipedia.org/wiki/Likert_scale. Dostopano: 09.09.2019.

[10] Maturity model. Dosegljivo: https://en.wikipedia.org/wiki/Maturity_model. Dostopano: 09.09.2019.

[11] Microsoft excel. Dosegljivo: https://products.office.com/en/excel, note = Dostopano: 09.09.2019.

[12] Slite. Dosegljivo: https://slite.com/. Dostopano: 08.09.2019.

[13] Sonarqube. Dosegljivo: https://www.sonarqube.org/. Dostopano: 08.09.2019.

[14] What is devops. Dosegljivo: https://aws.amazon.com/devops/what-is-devops/. Dostopano: 30.06.2019.

[15] Moira Alexander. Agile project management: A comprehensive guide. Dosegljivo: https://www.cio.com/article/3156998/agile-project-management-a-beginners-guide.html. Dostopano: 08.09.2019.

[16] John Allspaw and Paul Hammond. 10+ deploy per day. Dosegljivo: https://www.youtube.com/watch?v=LdOe18KhtT4, 2009. [Dostopano: 30. 06. 2019].

[17] Alexaxnder Boden, Gabriela Avram, Liam Bannon, and Volker Wulf. Knowledge management in distributed software development teams – does culture matter? *2009 Fourth IEEE International Conference on Global Software Engineering*, 2009.

[18] G Bou Ghantous and Asif Gill. Devops: Concepts, practices, tools, benefits and challenges. *PACIS2017*, 2017.

[19] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. Devops. *Ieee Software*, 33(3):94–100, 2016.

[20] Martin Fowler and Matthew Foemmel. Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, 122:14, 2006.

[21] Hillel Glazer, Jeff Dalton, David Anderson, Michael D Konrad, and Sandy Shrum. Cmmi or agile: why not embrace both! 2008.

[22] Michael Httermann. *DevOps for developers*. Apress, 2012.

[23] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer. What is devops?: A systematic mapping study on definitions and practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016*, page 12. ACM, 2016.

[24] Jerry Kilpatrick. Lean principles. *Utah Manufacturing Extension Partnership*, 68:1–5, 2003.

[25] Gene Kim, Kevin Behr, and George Spafford. *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win*. IT Revolution Press, 2013.

[26] Gene Kim, Jez Humble, Patrick Debois, and John Willis. *The DevOps Handbook: How to Create World-Class Agility, reliability and Security in Technology Organizations*. IT Revolution Press, 2016.

[27] Samer I Mohamed. Devops shifting software engineering strategy-value based perspective. *International Journal of Computer Engineering*, 17(2):51–57, 2015.

[28] Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, and Mike Amundsen. *Microservice architecture: aligning principles, practices, and culture*. "O'Reilly Media, Inc.", 2016.

[29] Fábio Oliveira, Tamar Eilam, Priya Nagpurkar, Canturk Isci, M Kalantar, W Segmuller, and E Snible. Delivering software with agility and quality in a cloud environment. *IBM Journal of Research and Development*, 60(2-3):10–1, 2016.

[30] Dan Packer. Continuous integration vs. continuous delivery vs. continuous deployment. Dosegljivo: `https://www.plutora.com/blog/continuous-integration-continuous-delivery-continuous-deployment`, note = Dostopano: 08.09.2019.

[31] Amol Patwardhan, Jon Kidd, Tiffany Urena, and Aishwarya Rajgopalan. Embracing agile methodology during devops developer internship program. *arXiv preprint arXiv:1607.01893*, 2016.

[32] Cynthia K. Riemenschneider, Bill C. Hardgrave, and Fred D. Davis. Explaining software developer acceptance of methodologies: a comparison of five theoretical models. *IEEE transactions on Software Engineering*, 28(12):1135–1145, 2002.

[33] Everett M Rogers. *Diffusion of innovations*. Simon and Schuster, 2010.

[34] Chuck Rossi. Rapid release at massive scale. Dosegljivo: `https://atscaleconference.com/videos/rapid-release-at-massive-scale/`, 2017. [Dostopano: 30. 06. 2019].

[35] Kirill Shirinkin. *Getting Started with Terraform*. Packt Publishing Ltd, 2017.

[36] Deepak Sinha. Drive business outcomes using microsoft azure devops. Dosegljivo: `https://www.techaheadcorp.com/blog/microsoft-azure-devops/`, 2019. [Dostopano: 09. 09. 2019].

[37] Tomislav Slijepčević. Primerjava orodij za upravljanje konfiguracij. Diplomska naloga, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2014.

[38] John Ferguson Smart. *Jenkins: The Definitive Guide: Continuous Integration for the Masses*. "O'Reilly Media, Inc.", 2011.

[39] CMMI Product Team. Cmmi for development, version 1.3. Technical report, 2010.

[40] Lenka Vašková. *Advanced Methods for Integration Testing*. PhD thesis, Masarykova univerzita, Fakulta informatiky, 2014.

[41] Damjan Vavpotič and Marko Bajec. An approach for concurrent evaluation of technical and social aspects of software development methodologies. *Information and software Technology*, 51(2):528, 545, 2009.

[42] Damjan Vavpotič and Tomaž Hovelja. Improving the evaluation of software development methodology adoption and its impact on enterprise performance. *Computer Science and Information Systems*, 9(1):165, 187, 2012.

[43] Abubaker Wahaballa, Osman Wahballa, Majdi Abdellatief, Hu Xiong, and Zhiguang Qin. Toward unified devops model. In *2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 211–214. IEEE, 2015.