UNIVERZA V MARIBORU
FAKULTETA ZA ELEKTROTEHNIKO,
RAČUNALNIŠTVO IN INFORMATIKO

Aljaž Heričko

UPORABA PROGRAMSKIH METRIK ZA NAPOVEDOVANJE TEŽAVNOSTI VZDRŽEVANJA KODE

Magistrsko delo

Maribor, september 2017

UNIVERZA V MARIBORU
FAKULTETA ZA ELEKTROTEHNIKO,
RAČUNALNIŠTVO IN INFORMATIKO

Aljaž Heričko

UPORABA PROGRAMSKIH METRIK ZA NAPOVEDOVANJE TEŽAVNOSTI VZDRŽEVANJA KODE

Magistrsko delo

Maribor, september 2017

UPORABA PROGRAMSKIH METRIK ZA NAPOVEDOVANJE TEŽAVNOSTI VZDRŽEVANJA KODE

Magistrsko delo

Študent:	Aljaž Heričko
Študijski program:	študijski program 2. stopnje
	Informatika in tehnologije komuniciranja
Mentor:	doc. dr. Boštjan Šumak
Lektor(ica):	Natalija Posavec, mag. prof. slovenskega jezika in knjiž.

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

Številka: E5021431
Datum in kraj: 01. 06. 2017, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Statut UM – UPB 11, Ur. l. RS, št. 44/2015)
izdajam

SKLEP O ZAKLJUČNEM DELU

1. **Aljažu Heričku**, študentu študijskega programa 2. stopnje MAG INFORMATIKA IN TEHNOLOGIJE KOMUNICIRANJA, se dovoljuje izdelati zaključno delo.

2. Tema zaključnega dela je pretežno s področja Inštituta za informatiko.

MENTOR: dr. Boštjan Šumak

3. Naslov zaključnega dela:
UPORABA PROGRAMSKIH METRIK ZA NAPOVEDOVANJE TEŽAVNOSTI VZDRŽEVANJA KODE

4. Naslov zaključnega dela v angleškem jeziku:
ANALYSIS OF METRICS FOR PREDICTING CODE MAINTAINABILITY DIFFICULTY

5. Zaključno delo je potrebno izdelati skladno z "Navodili za izdelavo zaključnega dela". Skladno s 7. členom *Pravilnika o postopku priprave in zagovora zaključnega dela na študijskih programih prve in druge stopnje Univerze v Mariboru,* je bilo odobreno podaljšanje roka za oddajo zaključnega dela do 03. 06. 2018. Zaključno delo študent odda v treh izvodih (dva trdo vezana izvoda in en v spiralo vezan izvod) ter en izvod elektronske verzije v referatu za študentske zadeve. Hkrati se odda tudi izjava mentor-ja/-ice (in morebitnega somentor-ja/-ice) o ustreznosti zaključnega dela.

Pravni pouk: Zoper ta sklep je možna pritožba na Senat članice v roku 10 delovnih dni od dneva prejema sklepa.

Dekan:
red. prof. dr. Borut Žalik

UNIVERZA V MARIBORU FAKULTETA ZA ELEKTROTEHNIKO, RAČUNALNIŠTVO IN INFORMATIKO 6

Obvestiti:
- kandidata,
- mentor-ja/-ico,
- odložiti v arhiv.

ZAHVALA

Zahvaljujem se mentorju, doc. dr. Boštjanu Šumaku, za vodenje in pomoč pri pripravi magistrskega dela.

Prav tako se zahvaljujem družini, ki me je v času izobraževanja podpirala in spodbujala.

Uporaba programskih metrik za napovedovanje težavnosti vzdrževanja kode

Ključne besede: programska oprema, kakovost, vzdrževalnost, metrike, indeks vzdrževalnosti

UDK: 004.4'2/'6(043.2)

Povzetek

Vzdrževanje je eno izmed najdražjih opravil v procesu razvoja programske opreme, po nekaterih študijah lahko predstavlja celo več kot polovico vseh stroškov projekta. V magistrskem delu smo opravili sistematičen pregled literature in raziskali metrike, ki se pogosto uporabljajo za oceno težavnosti vzdrževanja kode s pomočjo indeksa vzdrževalnosti. Analizirali in izbrali smo primerna orodja za pridobivanje metričnih vrednosti, potrebnih za izračun metrik vzdrževalnosti. Za štirideset odprtokodnih javanskih projektov smo izračunali vrednosti petih različic indeksa vzdrževalnosti in jih medsebojno primerjali. Skušali smo ugotoviti, katera izmed njih daje najboljši rezultat za napovedovanje na podlagi težavnosti izračuna ter doseganja primerljivih rezultatov s predhodno validiranim originalnim indeksom vzdrževalnosti. Ugotovili smo, da metrika, ki za delovanje potrebuje zgolj število vrstic, daje primerljive rezultate kot bistveno kompleksnejše metrike.

Analysis of metrics for predicting code maintainability difficulty

Key words: software, quality, maintainability, metrics, maintainability index

UDK: 004.4'2/'6(043.2)

Abstract

Maintainability is one of the more expensive steps in the process of developing software, according to some studies, it can represent over 50% of all the expenses of a project. In the thesis we did a systematic literature review, where we decided on further analyzing some metrics, which we found the best fit to use for predicting maintainability and we had the tools available, with which we could calculate the difficulty of maintenance. We decided on 5 metrics and we calculated their value on 40 chosen open source Java projects and we calculated the correlation between the values on different projects, as the effectiveness of one of the chosen metrics had already been validated, we could see how good the other metrics are to predict maintainability. Out of the five one metric stands out, as it only requires LOC metric, which is quite easy to get, to predict maintainability, while the result are comparable to the ones we get when using more complex methods to predict maintainability.

Kazalo

1 UVOD 1

1.1 Definicija in pomen vzdrževalnosti programske kode 2

1.2 Cilj in namen magistrskega dela 4

1.3 Struktura in potek izdelave naloge 4

2 SISTEMATIČNI PREGLED LITERATURE 6

2.1 Predstavitev poteka sistematičnega pregleda 6

2.1.1 Ključne besede 8

2.1.2 Vključitveni kriteriji 9

2.1.3 Evaluacijska forma 9

2.2 Ekstrakcija podatkov 10

2.2.1 Tip uporabljenih metrik 10

2.2.2 Nivo uporabljene metrike 10

2.2.3 Tip programskega jezika 11

2.2.4 Velikost projektov 11

2.2.5 Število analiziranih projektov: 11

2.2.6 Tip projektov 12

2.2.7 Dostopnost izvorne kode 12

2.2.8 Dosegljivost orodja 12

2.2.9 Analiza rezultatov 13

2.3 Izvedba sistematičnega pregleda 13

2.4 Izbrani članki 16

2.4.1 S1 – Uporaba metrik za napovedovanje vzdrževalnosti programske opreme 17

2.4.2 S2 – Razvoj in uporaba indeksa vzdrževalnosti za avtomatsko analizo izvorne kode 17

2.4.3 S3 – Napovedovanje vzdrževalnosti z uporabo objektno orientiranih metrik kompleksnosti zasnove 17

2.4.4 S4 – Ocenjevanje zmožnosti internih metrik za zgodnjo napoved težavnosti vzdrževanja skozi eksperimentiranje 18

2.4.5 S5 – Uporaba Bayesovih mrež za napovedovanje vzdrževalnosti programske opreme 19

2.4.6 S6 – Praktičen model za merjenje vzdrževalnosti 19

2.4.7 S7 – Napovedovanje vzdrževalnosti z uporabo tehnike MARS 20

2.4.8 S8 – Napovedovanje vzdrževalnosti odprokodnih projektov z metrikami za modeliranje 20
2.4.9 S9 – Objektno orientirane načrtovalske metrike in ogrodje MIMF 20
2.4.10 S10 – Indeks vzdrževalnosti programskih komponent z uporabo digrafa in matrik 21
2.4.11 S11 – Napovedovanje vzdrževalnosti z uporabo virtualne metode na osnovi relacijskih preslikav 21
2.4.12 S12 – Določitev indeksa vzdrževalnost za OO sisteme 21
2.4.13 S13 – Stroškovno naravnan model vzdrževalnosti programske opreme 21
2.4.14 S14 – Eksperimentalna študija strukturne kompleksnosti in strategije programerske skupine 22
2.4.15 S15 – Objektno orientirani razredi za napovedovanje vzdrževalnosti z uporabo internih meril o kakovosti kode 22
2.4.16 S16 – Računanje indeksa vzdrževalnosti programske opreme z uporabo števila vrstic 22
2.4.17 S17 – Pristop za izračun potrebnega dela za vzdrževanje na osnovi delitve 23
2.4.18 S18 – Empirična primerjava dveh skupin metrik za napovedovanje vzdrževalnosti v OO sistemih 23
2.4.19 S19 – Validacija učinkovitosti objektno orientiranih metrik za napovedovanje vzdrževalnosti 24
2.4.20 S20 – Napovedovanje vzdrževalnosti objektno orientiranih sistem z uporabo adaptivnih mrež 24
2.4.21 S21 – Uporaba indirektnih metrik sklopljenosti za napovedovanje vzdrževalnosti in testabilnosti paketa 24

2.5 Rezultati sistematičnega pregleda literature 25
2.5.1 Definicija vzdrževalnosti in uporaba standarda ISO 25
2.5.2 Predstavljene metrike in modeli za napovedovanje vzdrževalnosti 27
2.5.3 Uporabljene metrike 28
2.5.4 Množica podatkov 30
2.5.5 Uporabljena orodja 34
2.5.6 Pridobljeni rezultati 35

2.6 Izbira metrik in modelov za vzdrževalnost 36

2.7 Opis izbranih metrik 37
2.7.1 Osnovni indeks vzdrževalnosti (S1) 37
2.7.2 Izboljšan indeks vzdrževalnosti (S2) 37
2.7.3 Indeks vzdrževalnosti na osnovi LOC (S16) 38
2.7.4 Indeks vzdrževalnosti na osnovi nabora Martin metrik (S18) 38
2.7.5 Indeks vzdrževalnosti na osnovi nabora CK metrik (S18) 39

3 PREGLED ORODIJ ZA IZRAČUN METRIK ZA NAPOVEDOVANJE VZDRŽEVALNOSTI 40

3.1 Orodje JHawk 40

3.2 Orodje Sonar 41

3.3 Orodje JDepend 41

3.4 Orodje Ckjm 41

3.5 Orodje Classycle 42

3.6 Orodje Teamscale 42

3.7 JavaNCSS 42

3.8 Orodje CLOC 43

3.9 Orodje CCCC - C and C++ Code Counter 43

3.10 Orodje CMTJava 43

3.11 Rezultati vrednotenja in izbrano orodje 44

4 IZVEDBA EKSPERIMENTALNEGA DELA 46

4.1 Definicija in načrtovanje eksperimenta 46

4.1.1 Pearsonova korelacija 46

4.1.2 F-Test primerjava variance med dvema spremenljivkama 47

4.2 Opis in postopek izbire odprtokodnih projektov 48

4.3 Podatki o projektih 48

4.4 Izvedba eksperimenta 50

4.5 Statistična analiza zbranih rezultatov 54

4.6 Interpretacija rezultatov 56

4.7 Omejitve eksperimenta in tveganja za pravilnost 57

5 SKLEP 58

LITERATURA IN VIRI 59

Kazalo slik

Slika 1: Potek izvedbe magistrske naloge. 5
Slika 2: Potek sistematičnega pregleda literature. 7
Slika 3: Uporaba standarda ISO v člankih. 26
Slika 4: Število študij, ki uporabljajo določen tip metrik. 30
Slika 5: Število projektov. 32
Slika 6: Velikost projektov. 33
Slika 7: Tip projektov 33
Slika 8: Dostopnost izvorne kode projektov. 34
Slika 9: Analiza rezultatov. 35
Slika 10: Velikost odprtokodnih projektov. 49
Slika 11:Tip odprtokodnih projektov. 49

Kazalo tabel

Tabela 1: Ključne besede 8
Tabela 2: Rezultati izbrani članki 14
Tabela 3: Razlog neizbire. 15
Tabela 4: Izbrane študije 16
Tabela 5: Točkovanje tipa spremenljivke. 18
Tabela 6: SIG model. 19
Tabela 7: Definicija vzdrževalnosti in standard ISO. 25
Tabela 8: Predstavitev in opis metrik. 27
Tabela 9: Uporabljeni tipi metrik pri študijah. 29
Tabela 10: Število študij, ki uporablja določen tip metrik 29
Tabela 11: Uporabljeni projekti 31
Tabela 12: Uporabljena orodja v študijah. 34
Tabela 13: Izbira metrik. 36
Tabela 14: Izbira orodij. 44
Tabela 15: Izbrana orodja in metrike. 45
Tabela 16: Vrednosti metrik za izračun MI. 51
Tabela 17: Rezultati vrednosti metrik za izračun MIwCK. 52
Tabela 18: Vrednosti indeksov vzdrževalnosti. 53
Tabela 19: Rezultate koeficient korelacije. 54
Tabela 20: F-Test rezultati. 55

SEZNAM UPORABLJENIH KRATIC

API – vmesnik za namensko programiranje

AWT – Abstract Window Toolkit

Ca – število razredov v drugih paketih, ki so odvisni od razreda

CBO – število razredov, ki so vezani na določen razred

CC – ciklična kompeksnost

Ce – število razredov v drugih paketih, od katerih je odvisen razred

CK – Chidamber in Kemerer

CSA – povprečno število atributov razreda

CSMR – Konferenca o vzdrževalnost, reenžinirstvu in povratnem enžiniringu

CSO – povprečno število metod v razredu

GUI – grafični uporabniški vmesnik

HV – Halstead volumen

ISCM – Mednarodna konferenca na temo vzdrževanja in razvoja

ISO – mednarodna organizacija za standardizacijo

JPA – Java Persistence API

LOC – število vrstic kode

MARS – multivariate adaptive regression splines

MIMFW – Maintainability Index Metrics Framework

OO – objektno orientiran

OSAVG – povprečna kompleksnost metod

RFC – število različnih metod v razredu

SCAM – Konferenca o analizi izvorne kode in manipulacijami

SNOC – povprečno število otrok razreda

UML – poenoteni jezik modeliranja

1 UVOD

Programska koda se v današnjem svetu razvija v več iteracijah in se neprestano dopolnjuje. Slaba vzdrževalnost in tehnični dolg, ki izhaja iz tega, nas pri tem upočasnjuje in tako ovira nadaljnji razvoj. Vzdrževanje lahko tako predstavlja enega večjih problemov in stroškov pri razvoju programske opreme, vendar lahko težavnost in strošek vzdrževanja znižamo, če zagotovimo programsko kodo, ki je dobro strukturirana in enostavna za vzdrževanje. Pomembno je, da znamo pravilno oceniti in napovedati, kako težavno bo vzdrževanje določene programske opreme. Če ocenimo, da bo to prezahtevno, lahko pozornost namenimo izboljšanju kakovosti kode, saj s tem olajšamo vzdrževanje in dopolnjevanje programske opreme, s čimer prihranimo čas in denar.

Vzdrževalnost (angl. Maintainability) lahko pri določenih projektih predstavlja več kot 50 % vseh stroškov, ponekod celo več kot 80 % [1], ki ga imamo s projektom. Tako vsekakor predstavlja ključen del pri pripravi in uresničitvi projektov, na katerega moramo bili pozorni med vsemi fazami projekta. Trenutno se za ocenjevanje in izboljšanje težavnosti vzdrževanja kode v različnih fazah projekta uporabljajo različne metrike in pristopi. Tako se recimo v času razvoja pišejo razni testi, ki olajšajo testiranje sistema. Mi smo se osredotočili na metrike, ki se lahko izračunajo, ko že imamo napisan določen del izvorne kode programske opreme. Težavnost vzdrževanja se tako izračuna z uporabo metrik, ki temeljijo na interni strukturi kode. Najbolj pogosto uporabljena metrika je indeks vzdrževalnosti (angl. Maintainability index), katerega osnovna inačica je bila definirana leta 1994 [2], vendar kljub svoji starosti še vedno predstavlja dovolj dober izračun, da lahko napovemo vzdrževalnost kode. Indeks je bil zaradi svoje splošne uporabe tudi večkrat validiran [3], tako da je njegova učinkovitost napovedovanja znanstveno potrjena. Predlagane so bile tudi določene izboljšave tega indeksa, hkrati pa so različnih avtorji predstavili razne modele, ki jih lahko uporabimo za napoved vzdrževalnosti. Tako določeni avtorji za napovedovanje vzdrževalnosti uporabljajo razne tehnike s področja umetne inteligence, kot sta recimo metoda Naivni Bayes in metoda linearne regresije za izračun raznih parametrov, medtem ko drugi avtorji uporabljajo razne interne metrike, kot sta recimo povprečno število atributov razreda in povprečno število atributov razreda [4], ter s pomočjo teh metrik izračunajo vzdrževalnost. Obstaja torej veliko različnih načinov, kako izračunati vzdrževalnost kode.

V magistrskem delu smo, da bi bolje spoznali področje dela in različne metrike, ki temeljijo na indeksu kakovosti, opravili sistematičen pregled literature, s katerim smo analizirali, katere izmed teh programskih metrik so najpogosteje uporabljene, s katerimi dosežemo najboljše rezultate ter tako izbrali tiste, ki smo jih uporabili v empiričnem delu raziskave. V eksperimentalnem delu smo analizirali izbran nabor metrik vzdrževalnosti na odprtokodnih

projektih ter raziskali, če uporaba različnih metrik daje primerljive rezultate ali se vrednosti oz. napovedi težavnosti vzdrževanja med seboj znatno razlikujejo.

1.1 Definicija in pomen vzdrževalnosti programske kode

Standard ISO 9126, ki je nastal pod okriljem mednarodne organizacije za standardizacijo ISO (angl. International Standardization Organization), definira vzdrževanje kot težavnost, s katero spremenimo komponento oziroma programsko opremo, da odpravimo napako, izboljšamo delovanje ali vplivamo na ostale lastnosti programske opreme. Prav tako predstavlja spremembe, ki jih opravimo za prilagajanje programske opreme spremembam v okolju, kjer programska oprema deluje. Čeprav je standard star več kot petindvajset let, se definicija vzdrževalnosti kode ni veliko spremenila. V sedanjosti predstavlja vzdrževanje vsako dejavnost, ki je opravljena po tem, ko je bil produkt predan stranki v uporabo [5]. Pri tem moramo paziti na združljivost sistema z že izdano verzijo sistema, tako da ne povzročimo prekinitev v delovanju sistema, hkrati pa mora biti proces nadgradnje nezahteven za stranko [6]. Posledica vseh teh dejavnikov je, da je lahko vzdrževanje pri slabo zasnovanem sistemu izredno težko in zelo mučno ter problematično opravilo, zato moramo že pri razvoju programske opreme razmišljati, kako bi lahko olajšali vzdrževanje sistema po koncu njenega razvoja in opravljeni predaji.

Na slabo in težavno vzdrževanje vplivajo različni dejavniki. Najpogosteje je problem slaba kakovost kode, napake v izvorni kodi, neodkrite pomanjkljivosti, prezahtevnost tehnične rešitve, velikost sistemov in prevelika količina mrtve kode [7]. Informacijski sistemi s staranjem postajajo vedno večji in kompleksnejši ter v praksi večinoma tudi težje vzdrževalni [8], kar pomeni, da vsaka nadaljnja sprememba zahteva več dela, kakor bi bilo potrebno, če bi bil sistem enostaven za vzdrževanje. Posledica tega je predvsem zelo visok strošek ob odkritju napak oziroma pri željah stranke po spremembi delovanja sistema oziroma dodajanju funkcionalnosti.

V času zasnove in razvoja programske opreme se lahko določenim dejavnikom izognemo z uporabo raznih orodij, ki lahko preprečijo, da bi se takšne težave pojavile v naši kodi. Tako so podjetja začela z uporabo različnih sistemov, ki skrbijo za avtomatsko preverjanje kode in računanje težavnosti vzdrževanja.

Na začetku razvoja orodij za avtomatsko analizo težavnosti vzdrževanja kode so se avtorji upirali predvsem na standarde, ki so bili takrat priznani na področju kakovosti programske opreme. Na področju obstajata dva splošno priznana standarda oz. modela kakovosti, ISO 9126 in ISO 25010, ki je novejši in je nadomestil standard ISO 9126.

Prvi pomembnejši standard na področju kakovosti programske opreme je ISO 9126, ki je razdelil vzdrževanje programske kode na 6 delov. Cilj standarda je bil objektivno predstaviti določene dejavnike, ki vplivajo na kakovost programske kode. Standard je nastal leta 1991, ko je bil v ospredju problem neobstoja splošnega merila, kdaj bi kodo lahko opredeli kot slabo ali dobro vzdržljivo. Novonastali standard naj bi ta problem reševal. Dokončna verzija standarda je izšla leta 2001. Standard so sestavljale štiri komponente, in sicer model kakovosti, zunanje in notranje metrike ter metrike kakovosti pri uporabi. Pri ocenjevanju kakovosti kode je bil standard razdeljen na šest večjih značilnosti, vsaka je obsegala več podznačilnosti. Glavne značilnosti oz. področja kakovosti so [4]:

- *Funkcionalnost*, ki obsega informacije o ustreznosti, natančnosti, varnosti in funkcionalni uporabnosti.
- *Zanesljivost*, ki predstavlja zrelost kode, toleranco do napak, možnosti okrevanja po izpadu in doseganje zanesljivosti.
- *Uporabnost* predstavlja razumljivost, učljivost, upravljivost, privlačnost in doseganje uporabnosti.
- *Učinkovitost* združuje informacije o časovni zahtevnosti, uporabi virov in doseganju učinkovitosti.
- *Vzdrževalnost* združuje področja analize, spremenljivosti, stabilnosti in doseganja vzdrževalnosti.
- *Prenosljivost* predstavlja prilagodljivost, nestabilnost, sobivanje, nadomestljivost in doseganje prenosljivosti.

Vsako izmed manjših področij je nato natančneje razdeljeno na manjše enote, ki jih lahko bodisi verificiramo bodisi ocenimo v programski kodi. Za področje vzdrževanja kode je pomembna predvsem peta značilnost, kjer so predstavljeni določeni kriteriji vzdrževanja kode, vendar je za enostavnost vzdrževanja programske opreme pomembna celotna ustreznost programske kode tudi z ostalimi deli standarda. Ko je bil standard aktualen, je bilo na voljo več orodij, ki so omogočala samodejni izračun kakovosti in vzdrževalnosti z uporabo faktorjev, ki jih je predstavil standard.

Leta 2011 je ISO 9126 zamenjal posodobljen standard ISO 25010, ki je predstavil nove in dopolnil nekatere že obstoječe točke. Standard je razdeljen na osem delov, in sicer so razdelili nekatere točke iz standarda ISO 9126 oziroma jih drugače kategorizirali. Tako sta novi značilnosti funkcionalna primernost in učinkovitost delovanja (preimenovani iz funkcionalnost oz. učinkovitost), dodani pa sta bili kompatibilnost, ki je bila odstranjena iz prenosljivosti in je zdaj samostojna kategorija ter varnost. Ostale karakteristike so ostale podobne kot pri predhodnem standardu, le da so jim bile dodane oziroma odvzete določene podkategorije. Vzdrževalnost ima tako po novem standardu pet vidikov, in sicer modularnost (angl. Modularity), ponovno uporabnost (angl. Resuability), zmožnost analiziranja (angl. Analyzability), enostavnost spreminjanja (angl. Modifiability) in testiranja (angl. Testability). Kljub temu da naj bi standard nadomestil ISO 9126, je bil pri tem nekaj

časa neuspešen, saj se je v praksi izkazal za manj uporabnega kot nekateri drugi načini nadzorovanja vzdrževalnosti [9].

1.2 Cilj in namen magistrskega dela

Cilj magistrskega dela je ugotoviti, katere metrike in orodja se uporabljajo za napovedovanje težavnosti vzdrževanja programske kode s pomočjo indeksa kakovosti. Tako smo najprej ugotovili, katere metrike se lahko uporabijo za napovedovanje in nato glede na izbrane metrike poiskali orodja, ki jih lahko uporabimo za pridobivanje vrednosti teh metrik.

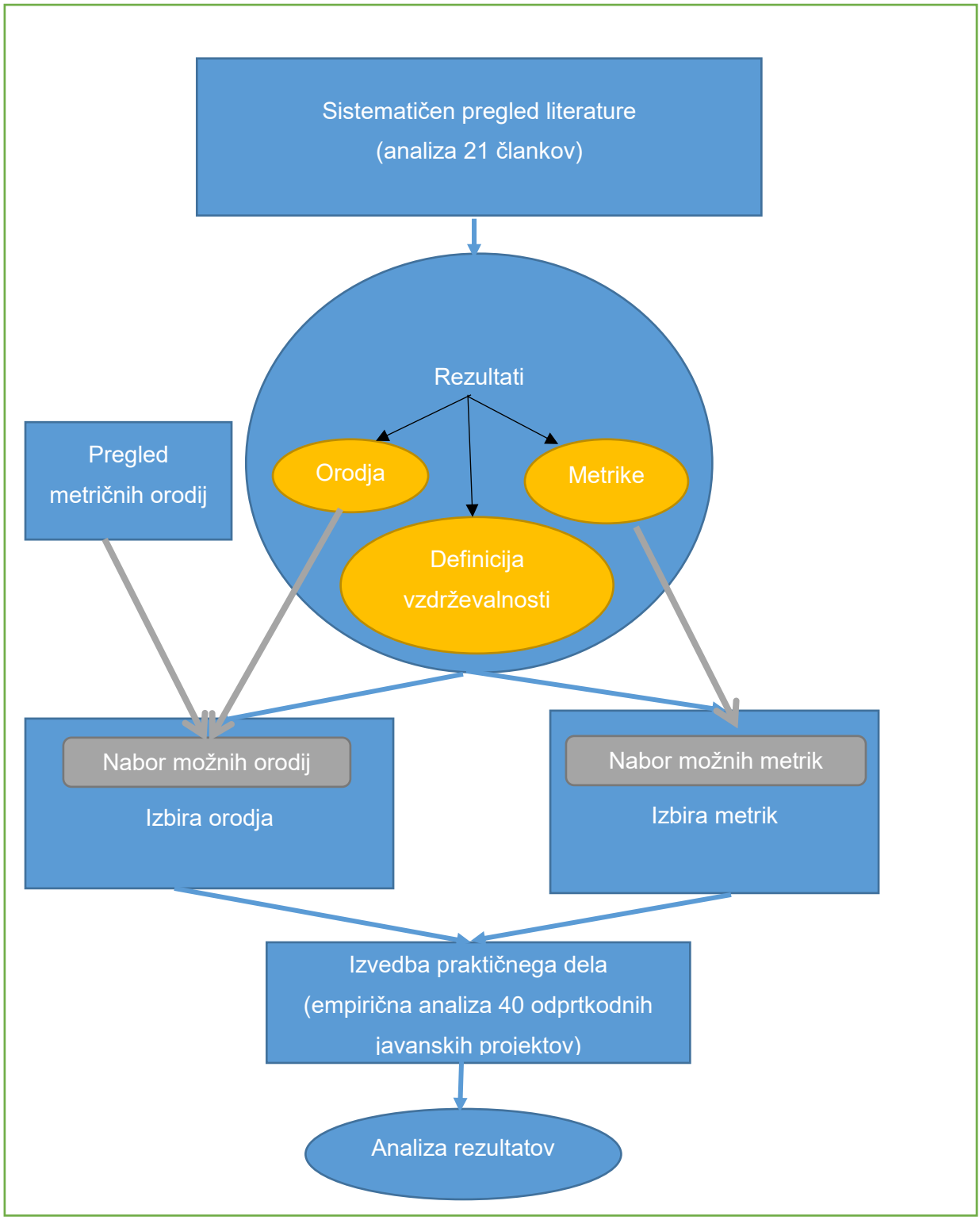
Ko smo imeli izbrane metrike, smo analizirali težavnost in zahtevnost izračuna vsake izmed metrik ter točnost rezultatov, ki jih posamezne metrike podajo. Glede na to, da je bila osnovna metrika na področju vzdrževanja oz. indeks vzdrževanja že validiran, smo primerjali ostale metrike na podlagi korelacije med gibanjem vrednosti validirane metrike z ostalimi. Cilj je bil odkriti, če obstaja metrika, ki bi omogočala primerljivo natančno napovedovanje vzdrževalnosti in bo njen izračun manj kompleksen ter posledično hitrejši kot izračun trenutno najbolj pogosto uporabljenega indeksa vzdrževalnosti.

1.3 Struktura in potek izdelave naloge

Magistrsko nalogo smo razdelili na tri dele, in sicer na izvedbo sistematičnega pregleda literature, analizo orodij in izvedbo eksperimentalnega dela. Prvi del bo izvedba sistematičnega pregleda literature, kjer smo analizirali obstoječa dela in stanje na področju napovedovanja vzdrževalnosti. Hkrati z analizo smo pridobili informacije o metrikah in modelih, ki so jih avtorji uporabili za napovedovanje vzdrževalnosti. Iz izbranih metrik in modelov smo izbrali tiste, ki so bili primerni za napovedovanje težavnosti. Nato smo poskusili najti orodja, ki podpirajo pridobivanje potrebnih podatkov za izračun izbranih metrik. V tem delu smo opisali in preizkusili orodja, ki smo jih našli med sistematičnim pregledom literature, prav tako pa smo jim dodali določena orodja, ki smo jih našli med iskanjem po spletu. V tretjem in zadnjem delu magistrske naloge smo izvedli eksperiment, kjer smo na odprtokodnih javanskih projektih izračunali vrednosti izbranih metrik in inačic indeksa vzdrževalnosti s pomočjo izbranih orodij iz prejšnjega dela. Rezultate smo nato medsebojno primerjali ter ugotavljali, ali metrike prikazujejo smiselne rezultate ter ali prihaja med njimi do odstopanj. Cilj je bil ugotoviti, katera izmed metrik je najpreprostejša in najhitrejša za izračun, vendar kljub temu še vedno daje dovolj dobre rezultate, da lahko z njeno uporabo napovemo, kakšna bo težavnost vzdrževanja aplikacije. Na sliki 1 je grafično prikazan potek izvedbe magistrske naloge.

Naloga je sestavljena iz petih poglavij. V prvem poglavju smo opisali področje in predstavili namen naloge. V drugem poglavju smo predstavili potek in rezultate sistematičnega pregleda literature. V tretjem poglavju smo analizirali orodja, ki jih lahko uporabimo za pridobivanje metričnih podatkov. V četrtem poglavju smo predstavili praktični del naloge,

kjer smo izbrane metrike in metode izračuna indeksa vzdrževalnosti, pridobljene s sistematičnim pregledom literature, uporabili na odprtokodnih projektih ter nato rezultate analizirali. V petem poglavju je sklep, v katerem smo strnili naše ugotovitve.



Slika 1: Potek izvedbe magistrske naloge.

2 SISTEMATIČNI PREGLED LITERATURE

Sistematični pregled literature je ena izmed pogosto uporabljenih raziskovalnih metod dela, s katero pridobimo osnovne podatke o tematiki ter izluščimo podatke za nadaljnjo uporabo. Glavna prednost sistematičnega pregleda literature je, da je sama metodologija dobro definirana in je natančno predpisana v literaturi [10][11]. Tako lahko bralec ve, kaj lahko od pregleda pričakuje in kako bo le-ta izveden.

2.1 Predstavitev poteka sistematičnega pregleda

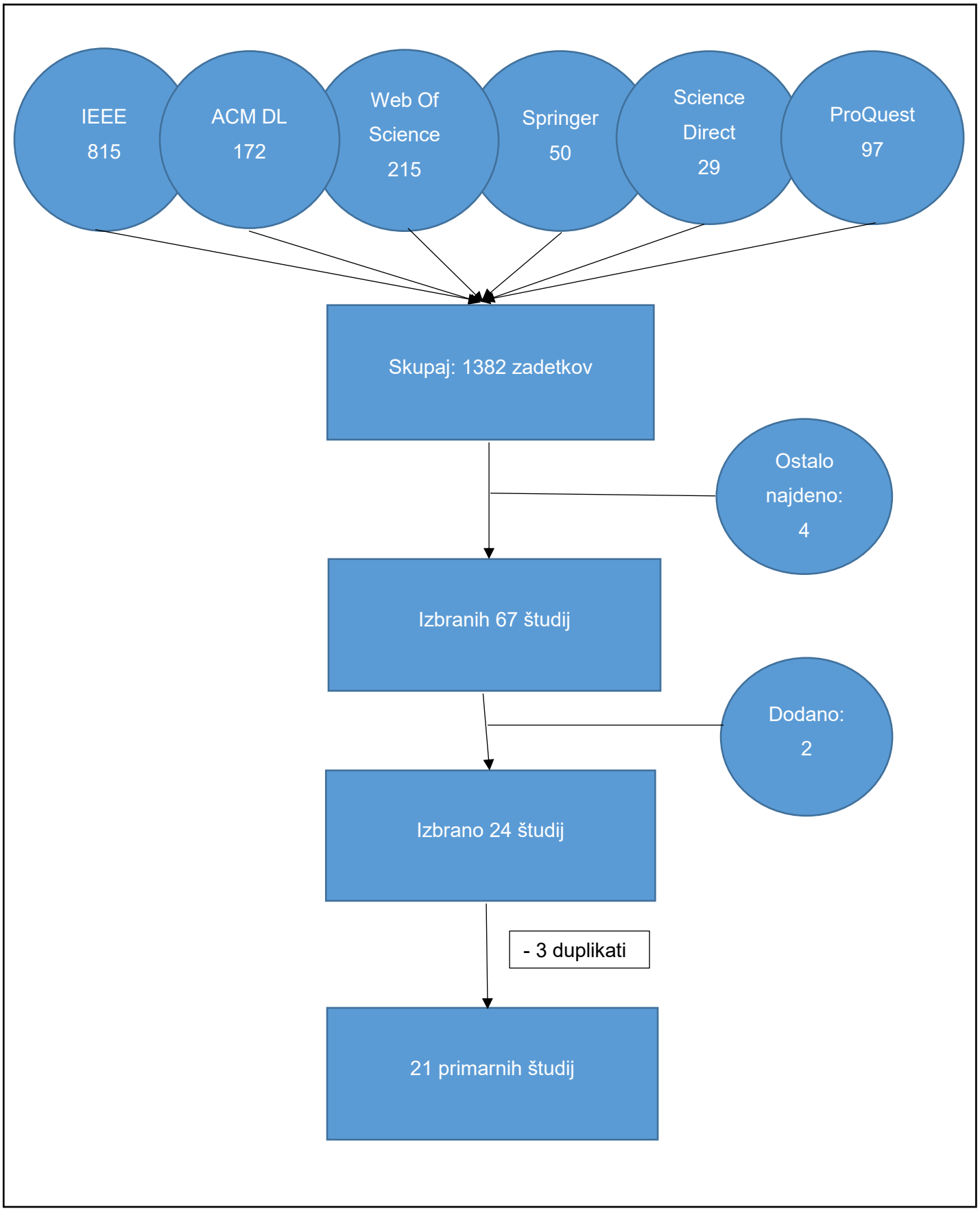
Sistematični pregled literature smo izvedli, tako kot je opisano v navodilih v [10], edina sprememba je bila, da smo zaradi prisotnosti le enega ocenjevalca določene dele prilagodili in tam, kjer naj bi sodelovalo pri ocenjevanju več oseb, odločitve temeljili zgolj na mnenju edinega ocenjevalca.

Kot prvo smo izbrali ključne besede in sestavili iskalni niz. Proces izbire ključnih besed je natančneje opisan v naslednjem poglavju. Kot naslednje smo definirali vključitvene kriterije za uporabo določenih študij.

Nadaljevali smo z izdelavo evaluacijske forme, ki je bila sestavljena s pomočjo petih raziskovalnih vprašanj. Forma je natančneje opisana v poglavju 2.2.

Izbrani iskalni niz smo nato uporabili na šestih bazah člankov, in sicer IEEE, ScienceDirect, ACM DL, Springer, WebOfScience in ProQuest. Članke, ki so bili med zadetki iskanja, smo nato na vsaki strani pregledali ter prvotno iz osnovnih podatkov o članku, ki so na voljo brez pregleda dokumenta članka, poskušali oceniti, ali zadostujejo vključitvenim kriterijem ter, če smo ugotovili, da jim zadostujejo, jih prenesli v naslednji krog izločanja. V tem delu smo dodali določene članke, ki niso vsebovali vseh želenih ključnih besed, vendar smo jih pri osnovnem pregledu tematike zasledili kot možne kandidate za končno analizo. Prav tako smo ročno pregledali nekatere najpomembnejše konference na področju vzdrževalnosti in izbrali članke, ki bi lahko bili uporabljeni pri sistematičnemu pregledu literature in niso vsebovali vseh izbranih ključnih besed. Pri tem smo pogledali članke, ki so se pojavili na konferencah ISCM (Mednarodna konferenca na temo vzdrževanja in razvoja), SCAM (Konferenca o analizi izvorne kode in vzdrževanju programske opreme) in CSMR (Konferenca o vzdrževanju in povratnem inženirstvu programske opreme).

V naslednjem krogu smo nato članke ponovno pregledali, in sicer tako, da smo analizirali tudi celotno besedilo člankov in tako pridobili boljšo predstavo o tem, ali je članek primeren za nadaljnjo analizo. Slika (glej Slika 2) prikazuje, kako je potekala izbira člankov in kako se je zmanjševalo njihovo število. Natančneje je proces razložen v kasnejšem poglavju.



Slika 2: Potek sistematičnega pregleda literature.

Kot naslednje smo izpolnili evaluacijsko formo in tako dobili rezultate za vsakega izmed člankov. Prav tako smo pregledali reference vsakega izmed del ter med njimi poskušali najti dela, ki bi jih lahko dodatno analizirali. To smo ponavljali, dokler nismo naleteli na smiselne reference in tako zaključili našo analizo. Rezultate smo opisali v sledečem poglavju, kjer smo jih razdelili na smiselne celote ter jih nato posamično podrobneje analizirali. Cilj sistematičnega pregleda je bil tudi izbira metodologij in metrik za nadaljnjo uporabo v eksperimentalnem delu, kar smo storili v zadnjem podpoglavju tega poglavja.

2.1.1 Ključne besede

Pri sistematičnem pregledu literature je treba izbrati ključne besede, s pomočjo katerih pridobimo članke za nadaljnjo obravnavo. Ključne besede smo razdelili na pet sklopov in sklope nato povezali z veznikom IN. V vsakem sklopu smo med ključnimi besedami uporabili besedo ALI, saj mora članek iz vsakega izmed sklopov vsebovati vsaj eno ključno besedo. Prvi izmed sklopov je sestavljen iz besed, ki vsebuje informacije o tem, nad čimer je bila pregledana vzdrževalnost. Drugi sklop predstavljata besedi vzdrževalnost ali vzdrževanje, ki jih mora članek vsekakor imeti. Tretji sklop pomaga pridobiti tiste članke, ki predstavijo določeno metodo, model ali metriko za pridobivanje vrednosti. V četrtem sklopu določimo, da morajo članki napovedati oziroma izračunati določeno vrednost. Ker je bilo število zadetkov previsoko, smo nato še dodali, da se mora članek sklicevati na objektno orientirano kodo. Kot smo že prej zapisali, smo besede v sklopu združili z veznikom ALI (ang. OR) ter sklope med seboj z besedo IN (ang. AND), kar določa, da mora besedilo vsebovati vsaj eno besedo iz vsakega sklopa. Prav tako smo določili, da smo pregledali le članke, kjer se ključne besede pojavijo v osnovnih informacijah članka. Dobili smo spodaj zapisan iskalni niz (glej Tabela 1: Ključne besede).

Tabela 1: Ključne besede

Skupina		Ključne besede
Skupina 1	Produkt	software OR code OR program OR software application OR application OR system
Skupina 2	Tema	maintenance OR maintainability
Skupina 3	Rezultat	metrics OR measure OR measurement OR index OR model
Skupina 4	Končna ocena	prediction OR estimation OR predict OR forecast OR estimate
Skupina 5	Paradigma programiranja	object OR oriented

2.1.2 Vključitveni kriteriji

Pri izbiri člankov smo si pomagali z vključitvenimi kriteriji, s pomočjo katerih smo določili, če določen članek ustreza našim zahtevam in je tako lahko uporabljen v nadaljnji analizi. Vključitveni kriteriji so:

- Članek mora biti recenziran in posledično objavljen na konferenci, znanstveni publikaciji ali na delavnici.
- Članek mora praktično prikazati pridobitev rezultatov ter ne sme podajati le teoretičnih zasnov.
- Metrike, opisane in analizirane v članku, morajo biti povezane z ocenjevanjem/vrednotenjem vzdrževalnosti programske opreme.

Publikacije, ki niso izpolnjevale teh treh pogojev, smo izločili iz sistematičnega pregleda literature.

2.1.3 Evaluacijska forma

V sklopu sistematičnega pregleda literature smo uporabili evaluacijsko formo, s pomočjo katere smo pridobili določene kategorije informacij iz različnih virov. V sklopu evaluacijske forme, ki je podana kot Priloga A, smo si zastavili pet raziskovalnih vprašanj, kjer je vsako vprašanje imelo tudi določeno število podvprašanj, na katera smo skušali z ekstrakcijo podatkov odgovoriti.

Raziskovalno vprašanje 1: Kako so avtorji definirali vzdrževalnost?

Raziskovalno vprašanje 1.1: Ali je bil pri definiciji vzdrževalnosti uporabljen standard ISO in katera verzija standarda?

Raziskovalno vprašanje 2: Kako se imenuje metrika za izračun vzdrževalnosti?

Raziskovalno vprašanje 2.1: Kakšna je formula metrike/metodologije?

Raziskovalno vprašanje 2.2: Na katerem nivoju deluje?

Raziskovalno vprašanje 2.3: Katere metrike in kateri tipi metrik so uporabljeni za izračun vzdrževalnosti?

Raziskovalno vprašanje 3: V katerem programskem jeziku so projekti, na katerih je bila metrika izračunana?

Raziskovalno vprašanje 3.1: Na kakšnih tipih projektov je bila metrika uporabljena?

Raziskovalno vprašanje 3.2: Ali je izvorna koda projektov dostopna?

Raziskovalno vprašanje 4: Ali je bilo uporabljeno orodje za pridobitev vrednosti?

Raziskovalno vprašanje 4.1: Ali je orodje prosto dostopno?

Raziskovalno vprašanje 5: Kakšni so rezultati metrike?

Raziskovalno vprašanje 5.1: Ali so bili rezultati primerjani glede na kakšne druge rezultate metrik za napovedovanje vzdrževalnosti oziroma ali so bili primerjani glede na vrednosti bodisi resničnega dela bodisi ocene dela programerjev?

Raziskovalno vprašanje 5.2: Kakšne so glavne ugotovitve članka?

2.2 Ekstrakcija podatkov

V poglavju smo opisali nabor vrednosti za vsako izmed vprašanj v formi za ekstrakcijo podatkov. Za vsako izmed izbranih publikacij smo najprej povzeli podatke o naslovu, avtorjih, letu izdaje, tipu objave ter ali je bila opravljena recenzija. Nato smo za vsako študijo preverili, če definira vzdrževalnost in če jo poveže s standardi ISO. V naslednjem koraku smo iz publikacije prevzeli formulo, ki jo uporablja za definicijo metrike ter uporabljene vrednosti za izračun metrike. Pri tem smo v nadaljnjih poglavjih tudi definirali vrednosti za tipe uporabljenih metrik. Prav tako smo definirali vrednosti, ki jih lahko zavzame, ter nivo, kjer je bila metrika uporabljena.

Kot naslednjo smo analizirali kodo projektov, na katerih je bila metrika uporabljena. Pregledali smo, kateri programski jezik je uporabljen v projektih, ali je njihova izvorna koda dostopna, za kakšno velikost projektov gre ter za kakšen tip projektov. Za lažjo primerjavo publikacij smo pri vsaki izmed teh vrednosti določili nabor vrednosti, ter kjer so bili podatki na voljo, uvrstili publikacijo v eno izmed skupin. Prav tako smo analizirali, če so avtorji uporabili katero izmed orodij za pridobivanje metričnih vrednosti. Kot zadnje smo analizirali, kako so avtorji publikacije analizirali lastno metodologijo ter ali so jo primerjali s katerimi preostalimi metrikami oziroma z realnim časom, potrebnim za izvedbo vzdrževanja. Pregled zaloge vrednosti vsake izmed kategorij je podan v nadaljevanju.

2.2.1 Tip uporabljenih metrik

Tipe uporabljenih metrik smo razdelili na štiri kategorije, in sicer glede na to, kakšne metrike se uporabijo v formuli, s pomočjo katere se izračuna težavnost vzdrževanja.

Nabor vrednosti:

- *tradicionalne*: metrike, ki se lahko uporabijo v vseh programerskih jezikih, niso odvisne od objektov in relacij med njimi,
- *objektno orientirane*: metrike, ki povejo nekaj o povezanosti med objekti in relacijah med njimi,
- *procesne*: metrike, ki se merijo izven izvorne kode in so del projekta, kot je recimo strošek dela,
- *načrtovalske*: metrike, ki se merijo v času priprave projektne dokumentacije, ko še ni na voljo izvorne kode, recimo v UML (angl. Unified Modeling Language) dokumentih.

2.2.2 Nivo uporabljene metrike

Metrike, ki jih avtorji predstavijo v svojih študijah, lahko delujejo na več različnih nivojih. To je za nas pomembno, saj tip metrike pove, kaj pravzaprav metrika analizira. Nižji kot je nivo metrike, več lahko povemo o vzdrževalnosti določenega dela in ugotovimo, kateri del bi bilo smiselno spremeniti oz. izboljšati.

Nabor vrednosti:

- na nivoju razreda: metrika deluje na nivoju razreda in tako pove o vzdrževalnosti razreda,
- na nivoju paketov: metrika deluje na nivoju paketa ter tako predstavlja, kakšna je vzdrževalnost paketa,
- na nivoju komponent: metrika lahko deluje na delu komponent oziroma modulov ter tako predstavlja vzdrževalnosti določene komponente,
- na nivoju aplikacije: metrike na tem nivoju predstavijo vzdrževalnost aplikacije kot celote ter tako lahko povejo le informacije o celotni aplikaciji.

2.2.3 Tip programskega jezika

Programske jezike lahko razdelimo v več skupin. Glede na tip programskega jezika, uporabljen v raziskavi, smo članek uvrstili v ustrezno kategorijo naslednjega nabora vrednosti:

- *objektno orientirani*: jeziki, ki za svoje delovanje uporabljajo objekte ter njihove metode,
- *proceduralni*: starejši tip jezika, ki za svoje delovanje ne uporablja objektov ter tako deluje z uporabo osnovnih programerskih idej,
- *jeziki za modeliranje*: jeziki za snovanje aplikacije, nimajo izvorne kode, vendar le diagrame, ki podajo določene informacije o strukturi programske kode.

2.2.4 Velikost projektov

Velikost projektov, ki so bili uporabljeni za napovedovanje vzdrževalnosti, poda določene informacije o tem, na kakšnih projektih so delovanje določene metrike preizkusili. Pri tem je predvsem pomembno, da je bila metrika preizkušena na večjih projektih in ne le majhnih. Prvi kriterij pri določitvi velikosti projekta je LOC (angl. Lines of Code) – število vrstic kode. V primeru, ko LOC ni na voljo, projekt razvrstimo na podlagi števila razredov.
Nabor vrednost:

- *majhni*: projekt ima manj kot 5000 LOC oziroma manj kot 100 razredov,
- *srednji*: projekt ima manj kot 50000 LOC oziroma manj kot 500 razredov,
- *veliki*: projekti, ki imajo več kot 50000 LOC oziroma več kot 500 razredov,
- *različno*: v to skupino spadajo vse študije, ki uporabijo projekte iz vsaj dveh izmed teh skupin.

2.2.5 Število analiziranih projektov:

Število projektov, podobno kot velikost projektov, je pomemben dejavnik pri ugotavljanju, ali je bilavdoločena študija validirana na dovolj velikem številu projektov. Pri tem je pomembno predvsem to, da je število projektov vsaj srednje vrednosti.

Nabor vrednosti:

- majhno: študije v tej skupini so uporabile en ali dva projekta za napovedovanje vzdrževalnosti,
- *srednje*: v tej skupini so vsi projektih, ki so uporabili več kot dva in manj kot deset projektov za validiranje lastne metode za napovedovanje vzdrževalnosti,
- *visoko*: uporabljeno število projektov je večje od 10.

2.2.6 Tip projektov

Zbirali smo tudi informacije o tem, kakšni projekti so bili uporabljeni, saj je prav tako pomembna tematika projektov. Pri tem so lahko vrednosti metrik pri različnih tipih projektov različne, saj je tudi struktura kode drugačna.

Nabor vrednosti:

- *administracijske aplikacije*: aplikacije, ki se uporabljajo za administracijo podatkov,
- *GUI aplikacije*: aplikacije, ki imajo grafičen vmesnik in kjer je fokus na grafičnem delu aplikacije,
- *knjižnice*: programi, ki se uporabijo znotraj drugih in se z njihovo vključitvijo v aplikacije olajša implementacija funkcionalnosti,
- *različno*: uporabljeni so bili projekti iz več vrst skupin,
- *ni podatka*: določene študije niso podale podatkov o tem, kakšen tip aplikacije je bil uporabljen.

2.2.7 Dostopnost izvorne kode

Dostopnost izvorne kode je pomembna predvsem zato, da lahko ugotovimo ponovljivost predstavljenih študij ter ali lahko izvorno kodo določene študije uporabimo v lastni eksperimentalni študiji.

Nabor vrednosti:

- *nedostopna*: izvorna koda projektov ni dostopna, morda zato, ker so projekti lastniški oziroma posledica tega, da je projekt tako star, da izvorna koda ne obstaja več,
- *delno dostopna*: v to skupino smo uvrstili študije, ki imajo določene projekte, kjer je izvorna koda dostopna in določene projekte, kjer izvorna koda ni dostopna,
- *dostopna*: izvorna koda vseh projektov, uporabljenih za izračun vzdrževalnosti, je dostopna ter jo lahko uporabimo v eksperimentalnem delu.

2.2.8 Dosegljivost orodja

Določene študije povedo, katera izmed orodij so uporabile za pomoč pri izračunu vrednosti za napovedovanje vzdrževalnosti. Pri tem je pomembno predvsem dejstvo, da smo v eksperimentalnem delu magistrskega dela potrebovali določeno orodje za izračun vzdrževalnosti ter lahko tako s pomočjo tega dela vprašanja najdemo določeno orodje, ki bi ga lahko uporabili v eksperimentalnem delu.

Nabor vrednosti:

- *dostopno*: orodje je prosto dostopno in za njegovo uporabo ni treba plačati. Večinoma bodo orodja v tej kategoriji iz odprtokodnih projektov in bo dostopna tudi njihova izvorna koda,
- *delno*: orodja v tej kategoriji bodo imela poizkusno verzijo, ki bo brezplačna, vendar ne bo podpirala vseh funkcionalnosti in bi za celotno verzijo bilo treba plačati,
- *plačljivo*: orodje je plačljivo ter nima na voljo nobene preizkusne verzije.

2.2.9 Analiza rezultatov

Avtorji so pri analizi rezultatov lahko primerjali vrednosti in validirali svoj rezultate na različne načine. Tako je možno, da so primerjali rezultate z realnimi spremembami v izvorni kodi oziroma s tem, kako se je izvorna koda spreminjala skozi čas.

Nabor vrednosti:

- *primerjava z drugi metrikami*: določene študije validirajo svojo tehniko napovedovanja vzdrževalnosti z uporabo drugih metrik, ki so bile predhodno validirane. Tako določene študije uporabijo recimo vrednost indeksa vzdrževalnosti za preverjanje, če njihova metoda deluje dobro;
- *primerjava z realnimi spremembami*: določene študije validirajo metriko tako, da so skozi čas spremljali spremembe na določeni programski opremi in tako dobili realne podatke o tem, koliko je bilo sprememb in so nato primerjali, če število sprememb ustreza napovedim;
- *primerjava med različnimi verzijami*: določene študije so vzele en projekt ter pogledale, kako se je spreminjala ta programska oprema ter nato na podlagi teh sprememb ugotovili, ali je njihova metrika primerna;
- *brez primerjave z ostalimi vrednostmi*: določene študije ne primerjajo dobljenih vrednosti na nobenega izmed načinov, temveč le predstavijo svoje rezultate in jih ne validirajo.

2.3 Izvedba sistematičnega pregleda

V tem poglavju smo zapisali postopek izvedbe sistematičnega pregleda literature. Natančneje smo zapisali število člankov, ki smo jih pregledali v določenem koraku ter pojasnili razloge, zakaj smo iz sistematičnega pregleda literature izločili določene članke. Kot prvo smo uporabili spodaj naveden iskalni niz za pridobitev rezultatov. Kako smo do iskalnega niza prišli, smo natančneje razložili v poglavju 2.1.1

(software OR code OR program OR software application OR application OR system)
AND (maintenance OR maintainability)
AND (metrics OR measure OR measurement OR index OR model)
AND (prediction OR estimation OR predict OR forecast OR estimate)
AND (object OR oriented)

V tabeli (glej Tabela 2) lahko vidimo število rezultatov, ki smo jih dobili pri uporabi ključnih besed v posamezni bazi.

Tabela 2: Rezultati izbrani članki.

Podatkovna baza	Zadetkov iskanja	Prva selekcija	Izbrano
IEEE	815	14	2
ProQuest	97	7	0
ACM DL	172	6	1
ScienceDirect	29	8	4
WebOfScience	215	24	7
Springer	50	1	1
Ostale	4	3	9
Vsota	1382	63	24

Vidimo lahko, da smo imeli po prvotnem vnosu ključnih besed v vsako izmed podatkovnih baz skupaj 1382 zadetkov na vseh straneh. Največ zadetkov je bilo na strani IEEE, kjer je bilo 815 zadetkov za iskani niz. Pri pregledu največjih konferenc na področju vzdrževalnosti programske opreme smo tem zadetkom dodali še 4 dela, ki smo jih našli na teh konferencah. Kot naslednje smo pregledali osnovne informacije vsakega izmed del, kot so naslov, povzetek, ključne besede ter kje je bilo določeno delo objavljeno. Pri izbiri, ali določen članek uvrstimo v nadaljnjo analizo, smo uporabili vključitvene kriterije, ki smo jih zapisali v 2.1.2.

Kot je razvidno iz tabele (glej Tabela 2), smo na podlagi vključitvenih kriterijev izbrali 63 člankov, ki bi potencialno ustrezali našim kriterijem oziroma na podlagi povzetka nismo uspeli ugotoviti, ali določeno dela ustreza kriterijem ali ne. Kot lahko tudi preberemo iz tabele, je bilo v tem koraku največ člankov iz baze WebOfScience, in sicer 24. V naslednjem koraku smo vse članke, ki so uspeli priti v naslednji krog, prebrali v celoti. Glede na prebrano smo lahko natančno sklepali, če določen članek ustreza našim kriterijem za vključitev v podrobnejšo analizo. Pri tem smo za vsak članek pregledali uporabljeno literaturo in če smo zasledili kakšno delo, ki bi lahko potencialno ustrezalo našim kriterijem, le-tega dodali med dela, ki smo jih v celoti prebrali in se glede na to odločili, če določeno delo uporabimo.

V tem koraku smo iz 63 potencialnih študij izbrali 24 primarnih študij, na katerih smo izvedli ekstrakcijo podatkov, podatke vnesli v evaluacijsko formo ter jih analizirali v naslednjih poglavjih. V tabeli (glej Tabela 3) lahko najdemo razloge, zakaj smo se odločili, da določeno delo zavrnemo. Kot lahko vidimo, smo pri analizi referenc posamičnih del naleteli na 9 dodatnih del, ki smo jih izbrali za natančnejšo analizo.

Tabela 3: Razlog neizbire.

Število zadetkov	Izbrano	Neizbrano	Brez rezultatov	Brez recenzije	Brez povezave z vzdrževalnostjo
IEEE	2	12	7	1	4
ProQuest	0	7	2	5	0
ACM DL	1	5	3	0	2
ScienceDirect	4	4	2	0	2
WebOfScience	7	17	6	0	11
Springer	1	0	0	0	0
Ostale	9	3	2	0	1
Skupaj	24	48	22	6	20

Kot lahko vidimo, smo v tem koraku izločili 48 del, ki niso ustrezala našim kriterijem glede vključitve določenega dela v zadnjo fazo sistematičnega pregleda literature. Kot je razvidno iz tabele, smo zaradi tega, ker niso imele povezave z vzdrževalnostjo, izločili 20 del, 24 del pa smo izločili, ker metrika, ki so jo podali avtorji, ni imela rezultatov in je bila le teoretična zasnova ideje, kako bi lahko izračunali vzdrževalnost. V prvi skupini so bila predvsem dela, ki so imela različne metode preoblikovanja kode, ki jih lahko uporabimo, vendar se niso dotaknila tematike vzdrževanja kode. Prav tako smo izločili 6 del, ki so bila le osnutek in še niso predstavljala končne verzije članka ali pa niso bila objavljena v nobeni literaturi in posledično niso bila ustrezno recenzirana.

Med 24 izbranimi deli lahko vidimo, da je največ člankov iz zbirke podatkov WebOfScience, od koder je 7 člankov. Sledi ScienceDirect s štirimi, IEEE z dvema, po en članek smo našli na ACM DL in Springer. Kot lahko opazimo v tabeli, smo iz referenc člankov našli še 6 dodatnih člankov, ki smo jih vključili v sistematičen pregled. Ker se med prvotnimi štiriindvajsetimi trije članki pojavijo dvakrat, imamo tako skupaj 21 člankov, ki ustrezajo kriterijem in smo jih podrobno analizirali. V naslednjem poglavju je podan seznam člankov, njihovih avtorjev ter njihove označbe. Sledi kratka predstavitev vsakega članka iz seznama. Predstavitev analize smo razdelili na pet večjih skupin. Kot prvo smo predstavili, kako so definirali vzdrževalnost, ali so pri tem uporabili standard ISO ter katero verzijo standarda. Nato smo predstavili metriko oziroma metodologijo, ki so jo uporabili za napoved vzdrževalnosti. Tretja skupina so interne metrike kode, ki so bile pri tem uporabljene, ter katera skupina metrik je največkrat uporabljena. V četrtem delu sledi predstavitev projektov, ki so jih avtorji uporabili za validacijo svojih metrik. Pri tem smo opisali, v katerem programskem jeziku so projekti ter ali je njihova izvorna koda dostopna. V zadnjem delu sistematičnega pregleda literature smo opisali, kako so validirali in analizirali rezultate svojih metrik.

2.4 Izbrani članki

V tem poglavju so zapisani izbrani članki, njihovi avtorji ter leto izdaje. Kot je zapisano v prejšnjem poglavju, smo izbrali 21 člankov.

Tabela 4: Izbrane študije.

Leto	ID	Avtor	Referenca
1994	S1	D. Coleman, D. Ash, B. Lowther et al.	[2]
1997	S2	Welker, Kurt D, Oman, Paul W in Atkinson, Gerald G	[12]
2003	S3	Bandi, Rajendra K, Vaishnavi, Vijay K in Turk, Daniel E	[13]
2005	S4	Bocco, Marcela Genero, Moody, Daniel L in Piattini, Mario	[14]
2006	S5	Koten, C Van, Gray, A R	[15]
2007	S6	Heitlager, Ilja, Kuipers, Tobias in Visser, Joos	[16]
2007	S7	Zhou, Yuming, Leung, Hareton	[17]
2008	S8	Zhou Yuming, Xu Baowen	[18]
2008	S9	Abd El-lateef, T, Yousef, A H in Ismail, M F	[19]
2010	S10	Upadhyay, Nitin, Deshapande, Bharat M, in Agrawal, Vishnu P.	[20]
2010	S11	Nair, T R Gopalakrishnan, Aravindh, Sri in Selvarani, R.	[21]
2011	S12	Kulwant Kaur, Dardeep Singh	[22]
2012	S13	Bakota, Tibor, Heged, Péter	[23]
2012	S14	Ramasubbu, Narayan, Kemerer, Chris F in Hong, Jeff	[24]
2013	S15	Jehad Al Dallal	[25]
2014	S16	Nahlah M. A. M. Najm	[26]
2014	S17	Alomari, Hakam W, Collard, Michael L in Maletic, Jonathan I	[27]
2014	S18	Madhwaraj K.G.	[4]
2015	S19	Kumar, Lov, Kumar, Debendra in Rath, Santanu Ku.	[28]
2015	S20	Swati Mishra, Arun Sharma	[29]
2016	S21	Almugrin, Saleh, Albattah, Waleed in Melton, Austin	[30]

V posameznem podpoglavju smo na kratko opisali vsakega izmed izbranih člankov ter strnili ugotovitve, ki so predstavljene v člankih.

2.4.1 S1 – Uporaba metrik za napovedovanje vzdrževalnosti programske opreme

Coleman, Ash et al. so prvi, ki uporabijo indeks vzdrževalnosti [2]. V članku je predstavljena pomembnost vzdrževalnosti. aAtorji razložijo postopek, kako so s pomočjo uporabe regresijskega testiranja pridobili formulo, ki jo je možno uporabiti za napovedovanje vzdrževalnosti. Formulo so nato dopolnili, saj je osnovna formula delovala nepravilno pod določenimi pogoji. Tako so prišli do prve formule za indeks vzdrževalnosti, ki je natančneje predstavljena v poglavju 2.6.1:

$$\mathrm{MI} = 171 - 3.42 * \ln(aveE) - 0.23 * aveV(g') - 16.2 * \ln(aveLOC) \qquad (2.1)$$

Formulo (glej Enačba 2.1Slika 2) so nato uporabili na sistemih znotraj podjetja HP na več projektih ter tako pridobili podatke o vrednostih metrike. Te so nato primerjali z realno težavnostjo vzdrževanja projektov in izračunali ter podali referenčne vrednosti metrike in kaj te pomenijo. Podane referenčne vrednosti so 85 točk in več, med 65 in 85 točk ter pod 65 točk. Vrednosti povedo, da je projekt z več kot 85 točkami enostavno vzdrževati, med 65 in 85 srednje težko in doseženih manj kot 65 točk pomeni, da je vzdrževanje projekta težko.

2.4.2 S2 – Razvoj in uporaba indeksa vzdrževalnosti za avtomatsko analizo izvorne kode

Članek [12] predstavi, kako poteka izračun z uporabo indeksa vzdrževalnosti ter različne formule, ki jih je možno najti in se imenujejo indeks vzdrževalnosti, saj se je definicija s časom posodabljala in spreminjala, kar tudi dokazujejo citirani članki. Welker, Oman et al. nato predstavijo novejšo formulo za izračun vzdrževalnosti (glej Enačba 2.2).

$$MI = 171 - 5.2 * \ln(HalV) - 0.23 * CC - 16.2 * \ln(aveLOC) \qquad (2.2)$$

Enačba je natančneje opisana v odseku 2.6.2. Avtorji navedejo več primerov, kjer je bila formula uporabljena za napovedovanje vzdrževalnosti in strnejo ugotovitve iz vsakega izmed primerov uporabe. Pri tem ugotovijo, da je nova formula primerna za uporabo v poslovnem svetu oz. za poslovne aplikacije.

2.4.3 S3 – Napovedovanje vzdrževalnosti z uporabo objektno orientiranih metrik kompleksnosti zasnove

Bandi, Vaishnavi et al. v članku [13] predstavijo tri možne načine, ki so lahko uporabljeni za izračun vzdrževalnosti v objektno orientiranih jezikih. Pri tem uporabijo dejstvo, da je težavnost uporabe spremenljivk različnih tipov različno težavna, zato predlagajo točkovanje spremenljivk, kot je predstavljeno v tabeli (glej Tabela 5). S pomočjo teh vrednosti razvijejo

tri različne metode napovedovanja vzdrževalnost, in sicer: nivo interakcij (IL – Interaction Level), velikost vmesnikov (IS – Interface Size) in kompleksnost operacij (OAC – Operation Argument Complexity).

Tabela 5: Točkovanje tipa spremenljivke.

Tip	Točkovanje
Boolean	0
Character ali Integer	1
Real	2
Array	3
Pointer	5
Record, Struct ali Object	6
File	10

Formule za metrike so zapisane v naslednjih enačbah (glej Enačba 2.3, 2.4, 2.5).

$$IL = K1 * (\check{s}tevilo\ interakcij) + K2 * (vsota\ to\check{c}k\ interakcij) \quad (2.3)$$

$$IS = K3 * (\check{s}tevilo\ parametrov) + K4 * (vsota\ to\check{c}k\ parametrov) \quad (2.4)$$

$$OAC = P(i), kjer\ je\ P(i) je\ \check{s}tevilo\ to\check{c}k\ vsakega\ parametra \quad (2.5)$$

Avtorji nato primerjajo rezultate realnih časov pri vzdrževanju kode z rezultati, kot bi jih napovedale metrike. Pri tem ugotovijo določeno korelacijo med metrikami in težavnostjo vzdrževanja, vendar bi za uporabo v poslovnem svetu bile potrebne nadaljnje raziskave.

2.4.4 S4 – Ocenjevanje zmožnosti internih metrik za zgodnjo napoved težavnosti vzdrževanja skozi eksperimentiranje

Bocco, Moody et al. se v članku [14] ukvarjajo z vzdrževalnostjo na osnovi razreda in sprememb v strukturi razredov v UML diagramih. Avtorji se dela lotijo tako, da določijo metrike, ki jih lahko izračunamo v UML diagramih, kot so recimo število razredov, število atributov itd. Avtorji preizkusijo določene hipoteze glede metrik. Hipoteze testirajo tako, da testnim osebam dodelijo 9 različnih projektov in od njih zahtevajo spremembo določenega dela strukture projekta. Glede na čas, potreben za spremembo, nato določijo, ali imajo metrike pozitivni ali negativni vpliv na vzdrževalnost. Njihove ugotovitve so, da imata predvsem dva parametra večji vpliv na težavnost vzdrževanja, in sicer število metod v razredih ter število asociacij med njimi. Avtorji tako ugotovijo, da bi se lahko že pri načrtovanju programske opreme, ko še ni izvorne kode, olajšala vzdrževalnost, če bi diagrame pripravili tako, da bi bile vrednosti omenjenih parametrov čim nižje.

2.4.5 S5 – Uporaba Bayesovih mrež za napovedovanje vzdrževalnosti programske opreme

Koten in Gray [15] uporabljata za napovedovanje vzdrževalnosti umetno inteligenco, in sicer uporabita Bayesove mreže. Tukaj lahko omenimo, da vsi članki, ki uporabljajo umetno inteligenco in smo na njih naleteli med sistematičnim pregledom literature, delujejo na podoben način in uporabijo isti podatkovni bazi za pridobivanje ter analizo rezultatov. Ti dve bazi sta QUES in UIMS. V tem članku avtorja uporabita Bayesovo formulo za izračun vzdrževalnosti, ki jo lahko najdemo v spodnji enačbi (glej Enačba 2.6). Rezultat je uspešno napovedovanje težavnosti na podlagi izbranih metrik glede na podatke o realnem trudu, ki je bil potreben za vzdrževanje obeh sistemov.

$$P(A|B) = \frac{P(B|A) * P(A)}{\mathrm{P(B)}} \tag{2.6}$$

2.4.6 S6 – Praktičen model za merjenje vzdrževalnosti

Heitlager, Kuipers in Visser delajo za podjetje, kjer so uporabljali indeks vzdrževalnosti za izračun vzdrževalnosti, vendar je pri tem prihajalo do težav. Razvili so svojo rešitev in jo opisali [16]. Primer je recimo, da indeks ne pove, zakaj pride do slabega rezultata oziroma zakaj so vrednosti v enačbi takšne, kot so. Zaradi teh problemov so se v njihovem podjetju odločili razviti nov model – SIG model vzdrževalnosti, ki bi rešil te probleme. Vzdrževalnost razdelijo na več delov po standardu ISO 9126, kjer se izvorna koda točkuje glede na vsakega izmed pogledov standarda. Pri tem se pri vsakem pogledu uporabi podobna tabela (glej Tabela 6). Za vsak del je natančno definirano, kakšno vrednost zavzame koda glede na to, v katero skupino jo uvrstimo.

Tabela 6: SIG model.

		Lastnosti izvorne kode					
		obseg	Kompleksnost enote	podvojenost	obseg enote	testi enote	
		++	--	-	-	0	
ISO 9126 kriteriji	možnost analize	x		x	x	x	0
	spremenljivost		x	x			-
	stabilnost						0
	testabilnost		x		x	x	-

V članku je omenjeno, da v podjetju uporabljajo natančnejšo formulo za izračun vzdrževalnosti, vendar ta ni zapisana v članku, tako da ne moremo točno vedeti, kako pridejo do izračuna končne vzdrževalnosti na podlagi svojega modela.

2.4.7 S7 – Napovedovanje vzdrževalnosti z uporabo tehnike MARS

Zhou in Leung izračunata vzdrževalnost programske kode z uporabo prilagodljivih regresijskih zlepkov z več spremenljivkami (angl. multivariate adaptive regression splines oziroma MARS), s pomočjo katerih pridobita rezultate za vzdrževalnost na podatkovnih bazah QUES in UIMS ter rezultate nato primerjata z ostalimi tehnikami, ki so že bile uporabljene na področju strojnega učenja in izračuna vzdrževalnosti.

Tehnika MARS je ena izmed pogosteje uporabljenih tehnik za izračun vzdrževalnosti, ker uporabimo že obstoječe podatke o vzdrževalnosti na izbranih bazah in potem pri določenih spremenljivkah spreminjamo vrednosti parametrov, da dobimo čim natančnejše rezultate. Te nato združimo in z uporabo parametrov ter spremenljivk napovemo vzdrževalnost novih različic programske kode.

Rezultati člankov dokazujejo, da je ta tehnika boljša in poda boljše rezultate kot druge tehnike na tem področju. Avtorja tako predlagata nadaljnje raziskovanje na področju uporabe tehnike MARS za napovedovanje vzdrževalnosti.

2.4.8 S8 – Napovedovanje vzdrževalnosti odprokodnih projektov z metrikami za modeliranje

Yuming in Baowen [18] izvedeta regresijsko logično testiranje za interne metrike in ugotavljata, ali lahko uporabimo izbrane metrike za izračun vzdrževalnosti. Avtorja ugotovita, da so določene metrike pri tem bolj uporabne kot druge. Te metrike so OSAVG – povprečna kompleksnost metod, CSO – povprečno število metod v razredu, CSA – povprečno število atributov v razredu in SNOC – povprečno število otrok razreda.

2.4.9 S9 – Objektno orientirane načrtovalske metrike in ogrodje MIMF

El-lateef, Yousef in Ismail v članku [19] opišejo delovanje metrik za objektno orientirano načrtovanje. Razvijejo Maintainability Index Metrics Framework (MIMFW), ki na osnovi podatkov izračuna končen indeks vzdrževalnosti glede na formulo, ki je kompleksnejša od osnovne. Formula je razdeljena na dva dela, Metrics Analyzer in PMI Calculator.

Pri tem uporabijo orodje Visual studio 2008 in lasten modul, ki izračuna vrednosti predlagane metrike. Za pridobitev rezultatov so pognali izračun na petih različnih verzijah določenega programa. Rezultati kažejo določeno stopnjo ujemanja z gibanjem vrednosti indeksa vzdrževalnosti in oceno ekspertov, tako da obstajajo možnosti za nadaljnje raziskave.

2.4.10 S10 – Indeks vzdrževalnosti programskih komponent z uporabo digrafa in matrik

Updahyay, Deshapaned in Agrawal v članku [20] razdelijo vzdrževalnost na več delov glede na informacije, ki so jih našli v literaturi. Nato vsakega izmed teh delov podrobneje razčlenijo ter jih ocenijo od 1 do 5 glede na njihovo pomembnost pri vzdrževalnosti. Pridobljene ocene vnesejo v matriko in iz nje izračunajo možne vrednosti za indeks vzdrževalnosti. Dobljenih rezultatov ne primerjajo z ničimer in prav tako ne povejo, kakšne mejne vrednosti bi predstavljale zadovoljivo stopnjo vzdrževalnosti.

2.4.11 S11 – Napovedovanje vzdrževalnosti z uporabo virtualne metode na osnovi relacijskih preslikav

Nair, Aravindh in Selvarani [21] izberejo določene metrike ter postavijo hipoteze, kako bodo spremembe metrik vplivale na vzdrževalnost. Teste nato izvedejo na dveh srednje velikih projektih in ugotavljajo, kakšen vpliv je imela določena metrika na vzdrževalnost. To izvedejo tako, da vrednosti metrik razvrstijo v različne skupine ter nato glede na vrednost skupine pogledajo, koliko skupin je imelo določeno število razredov, ki so bili spremenjeni. Pri tem s pomočjo pridobljenih rezultatov z linearno regresijo dobijo formulo za vsako izmed metrik, ki jih lahko uporabimo pri napovedovanju vzdrževalnosti. Ena izmed teh je zapisana v tem poglavju (glej Enačba 2.7).

$$\%\, Influence\ on\ Mantainability = \frac{\mathrm{MaC} * 100}{\mathrm{BoCM}} => Mainatainability\ Factor \qquad (2.7)$$

2.4.12 S12 – Določitev indeksa vzdrževalnost za OO sisteme

Kaur in Singh v [22] predlagata spremembo formule osnovnega indeksa vzdrževalnosti v takšno, ki bo uporabil objektno-orientirane metrike. Razlog je v tem, da metrike, uporabljene v prejšnjih verzijah indeksa vzdrževalnosti ne vsebujejo objektno orientiranih metrik. Metrike, uporabljene v osnovni formuli, torej LOC, CC (ciklomatična kompleksnost) in Halsteadov volumen so zamenjane z OO metrikami, kot sta recimo sklopljenost in vezljivost. Avtorja spremenita tudi nivo delovanja indeksa iz delovanja na nivoju aplikacije na nivo paketa. Rezultate nove formule nato primerjata s starim indeksom vzdrževalnosti in glede na rezultate ugotovita, da obstajajo možnosti za nadaljnje raziskave.

2.4.13 S13 – Stroškovno naravnan model vzdrževalnosti programske opreme

Bakota in Heged v članku [23] definirata nov pristop k ocenjevanju vzdrževalnosti na osnovi ideje iz termodinamike, kjer se meri stopnja izstopanja iz sistema. Za izračun vzdrževalnosti uporabita formulo (glej Enačba 2.8), kjer q predstavlja število sprememb, c predstavlja strošek v določenem trenutku, medtem ko je k konstanta za pretvorbo iz metrike v relativno vrednosti.

$$M\ (t) = e^{-\frac{q}{k}*c(t)} \tag{2.8}$$

Delovanje metrike preverita na petih projektih, natančneje treh lastnih in dveh odprtokodnih. Rezultati kažejo na smiselno zasnovo metrike ter uspešno napovedovanje vzdrževalnosti. Metrika tudi omogoča napovedovanje stroškov, ki jih bomo imeli z določenim projektom v prihodnosti.

2.4.14 S14 – Eksperimentalna študija strukturne kompleksnosti in strategije programerske skupine

Ramashubu, Kemerer in Hong v članku [24] opišejo takratno aktualno stanje pri uporabi OO metrik za napovedovanje vzdrževalnosti. Na podlagi teh metrik izpeljejo ogrodje, imenovano Distributed Cognition Framework in postavijo določene hipoteze glede delovanja metrik. Prav tako glede na tip dela eksperiment razdelijo na dva dela, in sicer na individualno delo ter delo v skupini.

Nato izvedejo eksperimentalno študijo, kjer poizkušajo potrditi oziroma zavreči postavljene hipoteze. Pri študiji so izbrali en javanski projekt, na katerem so pridobili podatke o realnem trudu, ki je bil potreben za vzdrževanje. Iz pridobljenih podatkov so skušali najti povezavo med OO metrikami in trudom. Pri tem ugotovijo, da je, razen v primeru visoke vezljivosti in nizke sklopljenosti, v vseh ostalih primerih vzdrževanje v skupini lažje in doseže boljši rezultat kakor samostojno delo.

2.4.15 S15 – Objektno orientirani razredi za napovedovanje vzdrževalnosti z uporabo internih meril o kakovosti kode

Dallal [25] uporabi interne metrike za merjenje vrednosti ter skuša ugotoviti, katere izmed metrik se lahko uporabijo za napovedovanje vzdrževalnosti. Izračunane vrednosti nato primerja z realnimi rezultati, ki so pridobljeni na podlagi števila spremenjenih vrstic. Iz pridobljenih rezultatov izpelje določene sklepe, ki lahko razvijalcu pomagajo pri izboljšanju vzdrževalnosti. Avtorjevi predlogi so recimo nižje število razredov ter manjša sklopljenost in večja vezljivosti med razredi.

2.4.16 S16 – Računanje indeksa vzdrževalnosti programske opreme z uporabo števila vrstic

Najm [26] predlaga poenostavitev osnovnega MI le z uporabo vrednosti števila vrstic kode (LOC) za pridobitev vseh potrebnih podatkov za izračun vzdrževalnosti. Pri tem za zamenjavo ostalih dveh metrik v formuli, ki sta CC in Halstead, uporabi lastni formuli, ki pri izračunu namesto klasičnih formul za izračun vrednosti uporabi le informacije o številu vrstic.

Nato avtor svoj nov algoritem primerja z že obstoječim na podlagi šestih projektov, kjer ugotovi izredno podobnost rezultatov na vseh šestih projektih ter tako s poenostavitvijo formule pride do podobnih rezultatov, kakor jih dobimo z osnovnim indeksom vzdrževalnosti.

2.4.17 S17 – Pristop za izračun potrebnega dela za vzdrževanje na osnovi delitve

Alomari, Collard et al. predlagajo nov pristop k reševanju problema napovedovanja vzdrževalnosti [27]. Predlagajo način, poimenovan Slice-Based pristop, ki program razdeli na več delov in jih posamično analizira. V namen uporabijo za vsak del svoje spremenljivke, kot je recimo njihova velikost. Za izračun vzdrževalnosti so potrebni podatki o prejšnjih verzijah, saj se vzdrževalnost izračuna z uporabo podatkov o spremembah. Spodaj je podana formula (glej Enačba 2.9), ki jo avtorji uporabijo za izračun vzdrževalnosti.

$$E_{slice} = c1 + c2\ (sliceSize) + c3\ (hastSize) + c4\ (sCoverage) \tag{2.9}$$

Avtorji nato izvedejo analizo, s katero ugotavljajo, ali je predlagan pristop primeren. Pri tem si pomagajo z lastno razvito metriko za izračun težavnosti vzdrževanja na osnovi števila vrstic kode ter števila sprememb med verzijami. Analizo izvedejo nad projektom Linux Kernel, saj pri odprtokodnih projektih nimajo podatkov o spremembah. Rezultati kažejo na to, da je njihov pristop dober in lahko služi kot osnova za nadaljnje raziskave pri napovedovanju vzdrževalnosti.

2.4.18 S18 – Empirična primerjava dveh skupin metrik za napovedovanje vzdrževalnosti v OO sistemih

Članek [4] predstavi temo vzdrževalnosti programske opreme in različnih metrik, ki so jih predstavili različni avtorji in jih lahko uporabimo pri analizi kode. Pri tem se avtor osredotoči na dve skupini metrik, in sicer Martin metrike in CK (Chidamber in Kemerer) metrike. Madhwaray predstavi obe skupini, posamezne metrike, ki jo sestavljajo, ter kaj te metrike pomenijo. S pomočjo indeksa vzdrževalnosti z uporabo sistema več spremenljivk in prej omenjenih metrik, ki sestavljajo določeno skupino, poizkuša izpeljati formulo, ki bi jo lahko uporabili pri izračunu vzdrževalnosti kode na podlagi obeh skupin. Rezultat sta naslednji formuli:

$$MI = 55.855 - 13.678\ (D) - 2.577\ (Ca) - 0.154\ (CC) + 0.938\ (Ce) \tag{2.10}$$

V formuli (glej Enačba 2.10) *D* predstavlja razdaljo paketa od idealnega izraza, *Ca* predstavlja število razredov v drugih paketih, ki so odvisni od razreda, *Ce* predstavlja število razredov v drugih paketih, od katerih je odvisen razred, in *CC* predstavlja število razredov.

$$MI = 42.007 - 1.181\ (CBO) + 0.437\ (RFC) \quad (2.11)$$

V formuli (glej Enačba 2.11) *CBO* predstavlja število razredov, ki so vezani na določen razred, spremenljivka *RFC* (število različnih metod v razredu) pa pomeni število različnih metod, ki jih lahko izvede določen razred, ko dobi sporočilo. Rezultati, ki jih avtor pridobi na podlagi razvitih formul, predstavljajo dokaj dober približek in nadomestek za indeks vzdrževalnosti.

2.4.19 S19 – Validacija učinkovitosti objektno orientiranih metrik za napovedovanje vzdrževalnosti

Kumar, Rath et al. v članku [28] opravijo analizo trenutno obstoječih metod, ki uporabljajo umetno inteligenco, in OO metrik za napovedovanje vzdrževalnosti. S pomočjo pridobljenega znanja in ugotovljenih pomanjkljivosti obstoječih pristopov predlagajo nov pristop – Neuro-GA pristop. Ta pristop uporabi genetski algoritem, ki v času učenja spreminja uteži za določene metrike, dokler ni dosežen optimalen rezultat.

Avtorji lasten algoritem primerjajo z ostalimi algoritmi na področju umetne inteligence, kot so Bayesove mreže, razni regresijski in večnivojski modeli. Pri tem pridobijo boljše rezultate, kot jih imajo ostali algoritmi.

2.4.20 S20 – Napovedovanje vzdrževalnosti objektno orientiranih sistem z uporabo adaptivnih mrež

Mishra [29] izvede analizo literature, kjer ugotovi, da obstoječi sistemi na področju napovedovanja vzdrževalnosti z mehko (angl. Fuzzy) metodo uporabijo deset internih metrik za izračun in napoved vzdrževalnosti. Avtorjeva ideja je poenostaviti ta sistem s tem, da bi bilo namesto desetih metrik uporabljenih le pet, in sicer DIT, NOC, MPC, RFC in SIZE2.

Avtor rezultate primerja z rezultati, ki bi jih dobil, če bi uporabil vseh 10 metrik. Rezultati kažejo, da se nov algoritem odreže za 21 % slabše, kot obstoječe verzije in bi lahko bil v določenih primerih uporaben, saj je poenostavljen in deluje hitreje kot prejšnje verzije.

2.4.21 S21 – Uporaba indirektnih metrik sklopljenosti za napovedovanje vzdrževalnosti in testabilnosti paketa

Almugrin, Albattah et al. v članku [30] predlagajo nove metrike, ki bi se lahko uporabile pri napovedovanju vzdrževalnosti. Med temi so *InRank*, *OutRank* in *I(A)*. V formulah (glej Enačba 2.12, 2.13, 2.14) vidimo način izračuna vrednosti teh metrik.

$$InRank(A) = (1-d) + d * \left(\frac{InRank(T1)}{C(T1)} + \cdots + \frac{InRank(Tn)}{C(Tn)}\right) \quad (2.12)$$

$$OutRank(A) = (1-d) + d * \left(OutRank(P_1) + \cdots + OutRank(P_n)\right) \quad (2.13)$$

$$I(A) = \frac{OutRank(A)}{OutRank(A) + InRank(A)} \quad (2.14)$$

Te metrike nato testirajo na več javanskih odprtokodnih projektih in primerjajo z realnimi spremembami. Pri tem ugotovijo določeno stopnjo korelacije med predlaganimi metrikami in realnim stanjem, koliko sprememb je bilo v dokumentu, kar pomeni, da lahko predlagane metrike uporabimo za napovedovanje vzdrževalnosti.

2.5 Rezultati sistematičnega pregleda literature

2.5.1 Definicija vzdrževalnosti in uporaba standarda ISO

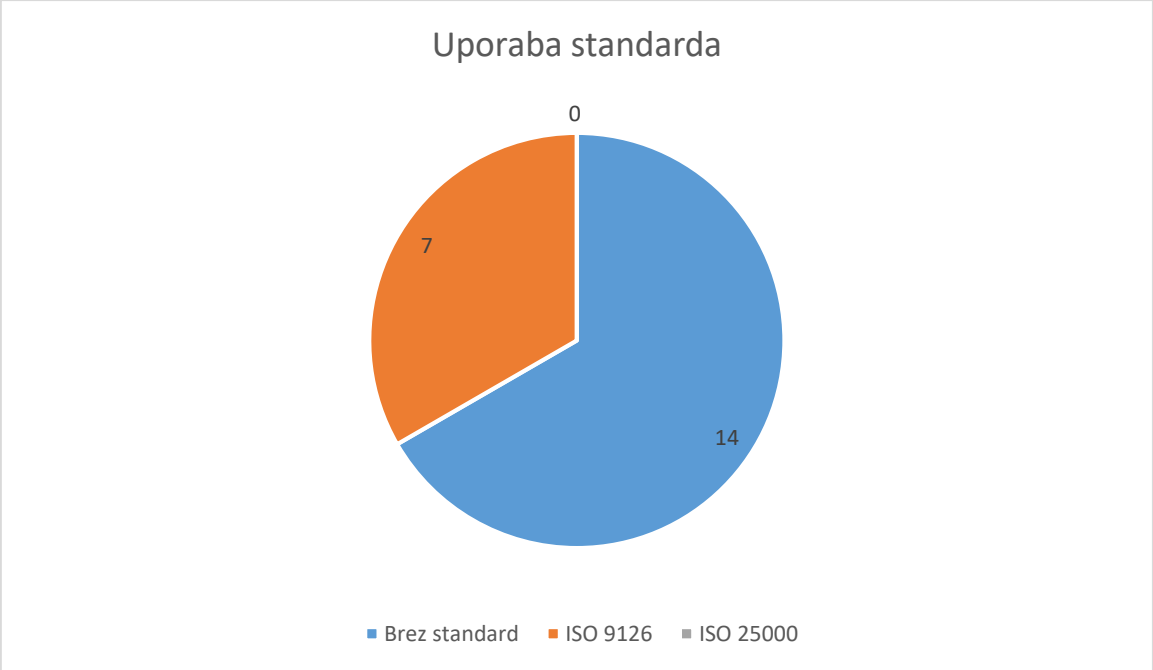
V tem poglavju bomo odgovorili na raziskovalno vprašanje 1 in 1.1. Najprej smo iz vseh izbranih študij vzeli definicijo vzdrževalnosti, kjer je le-ta opredeljena. Kot naslednje smo pogledali, ali je bil pri definiciji uporabljen standard ISO ter katera verzija standarda je bila uporabljena, saj kot smo že v uvodu napisali, je bil leta 2001 splošno priznan ISO 9126, vendar je bil leta 2011 izdan standard ISO 25010, ki bi ga novejši članki morali uporabljati. V tabelo (glej Tabela 7) smo vključili tudi tiste članke, ki ne definirajo vzdrževalnosti, vendar kljub temu uporabijo standard ISO pri definiciji metrike.

Tabela 7: Definicija vzdrževalnosti in standard ISO.

Članek	Definicija vzdrževalnosti	Uporaba ISO? Verzija?
S2	Vzdrževalnost je težavnost, s katero spremenimo komponento oziroma programsko opremo, da odpravimo napako, izboljšamo delovanje ali ostale lastnosti, oziroma prilagajanja na spremenjeno okolje.	Ne.
S4	Zmožnost sistema, da ga spremenimo.	Da, ISO 9126.
S5	Vzdrževalnost je število sprememb v kodi, ki so bile izvedene v času vzdrževanja.	Ne.
S6	Brez definicije.	Da, ISO 9126.
S10	Vzdrževanje je ena izmed lastnosti zasnove programske opreme in je zelo pomembna med delovanjem sistema.	Da, ISO 9126.
S11	Vzdrževalnost je težavnost pri spreminjanju oziroma nadgradnji programske opreme.	Ne.
S12	Vzdrževalnost je aktivnost popravljanja napak v sistemu, prilagajanje sistema na različna okolja in dodajanje ali spreminjanje funkcionalnosti sistema.	Ne.
S13	Vzdrževalnost je definirana kot trud ali delo (oziroma strošek), ki je potrebno za izvedbo spremembe v sistemu.	Da, ISO 9126.

S15	Težavnost, s katero spremenimo komponento oziroma programsko opremo.	Ne.
S16	Vzdrževalnost je atribut, ki je odvisen od težavnosti razumevanja programske opreme.	Da, ISO 9126.
S18	Težavnost, s katero spremenimo komponento oziroma programsko opremo, da odpravimo napako, izboljšamo delovanje ali ostale lastnosti, oziroma prilagajanja na spremenjeno okolje.	Ne.
S19	Zmožnost spremembe programske opreme vključujoč izboljšave oziroma prilagajanja, popravila opreme v uporabi in spreminjanje zahtev ter funkcionalnosti sistema.	Da, ISO 9126.
S20	Proces spreminjanja programske opreme, ki je že bila dostavljena uporabniku.	Ne.
S21	Težavnost, s katero spremenimo komponento oziroma programsko opremo, da odpravimo napako, izboljšamo delovanje ali ostale lastnosti, oziroma prilagajanja na spremenjeno okolje.	Da, ISO 9126.

V tabeli lahko opazimo, da so si definicije vzdrževalnosti med seboj precej podobne. Večinoma gre za težavnost, s katero lahko spremenimo sistem, natančneje definirano, ga adaptiramo ali izboljšamo, izvedemo popravke oziroma ga priredimo tako, da izpolnjuje določene zahteve ali specifikacije. Hitro lahko opazimo, da kar veliko člankov niti ne definira natančno, kaj vzdrževalnost je. Prav tako lahko ugotovimo, da le malo člankov pri definiciji uporablja standard ISO.



Slika 3: Uporaba standarda ISO v člankih.

Kot lahko vidimo (glej Slika 3), večina člankov ne uporabi standarda. Prav tako je zanimiva ugotovitev, da nobena izmed izbranih študij ne uporablja standarda ISO 25010, tudi članki

izdani po njegovi vpeljavi, torej po letu 2011, ki bi morali vsekakor pri definiciji in napovedovanju vzdrževalnosti uporabljati najnovejši standard ISO 25010.

2.5.2 Predstavljene metrike in modeli za napovedovanje vzdrževalnosti

V tem poglavju smo zapisali najpomembnejšo formulo in nivo delovanja, na katerem deluje določena metrika. Uporabili smo možne vrednosti, ki so zapisane pri formi za ekstrakcijo podatkov. Tako smo odgovorili na raziskovalni vprašanji 2, 2.1 in 2.2.

Tabela 8: Predstavitev in opis metrik.

Članek	Formula metrik	Nivo delovanja
S1	$\mathrm{MI} = 171 - 3.42 * \ln(aveE) - 0.23 * aveV(g') - 16.2 * \ln(aveLOC)$	Komponenta
S2	$MI = 171 - 5.2 * \ln(HalV) - 0.23 * CC - 16.2 * \ln(aveLOC)$	Komponenta
S3	$IL = K1 * (\text{št.}interakcij) + K2 * (vsota\ interakcij)$ $IS = K3 * (\text{št.}parametrov) + K4 * (vsota\ parametrov)$ $OAC = P(i), kjer\ je\ P(i)\ je\ \text{št.točk}\ vsakega\ parametra$	Razred
S4	$\text{Čas}\ vzdrževanja = 2.07 + (0.05 * \text{Število}\ metod)$	Komponenta in aplikacija
S5	$P(A\|B) = \frac{P(B\|A) * P(A)}{\mathrm{P(B)}}$	Razred
S6	Ni enačbe	Aplikacija
S7	$y = c_0 + \sum_{m=1}^{M} c_m \prod_{k=1}^{K_m} b_{km}(x_{v(k,m)})$	Razred
S8	Ni enačbe	Aplikacija
S9	$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{tc} M_a(C_i)}$ $PF = \frac{\sum_{i=1}^{TC} M_O(C_i)}{\sum_{i=1}^{tc}[M_n(C_i) x DCC_i]}$	Aplikacija
S10	$VPF - m = g(M_i, m_{ij}^2, m_{ij}m_{jk}m_{ki}\ etc.)$	Komponenta
S11	$\%\ Influence = \frac{\mathrm{MaC} * 100}{\mathrm{BoCM}} => Mainatainability\ Factor$	Razred
S12	$MIP = 171 - 0.23 * \ln(S) - 5.2 * CC - 16.2 * \ln(NC)$	Paket

S13	$M\ (t) = e^{-\frac{q}{k} * c(t)}$	Aplikacija
S14	Ni enačbe	Aplikacija
S15	$\pi(X) = \frac{1}{1 + e^{-(c_0 + c_1 x)}}$	Razred
S16	$MI = 171 - 5.2 * \ln(HV) - 0.23 * CC - 16.2 * \ln(LOC)$	Aplikacija
S17	$E_{slice} = c1 + c2\ (sliceSize) + c3\ (hastSize) + c4\ (sCoverage)$	Aplikacija
S18	$MI = 55.855 - 13.678\ (D) - 2.577\ (Ca) - 0.154\ (CC) + 0.938\ (Ce)$ $MI = 42.007 - 1.181\ (CBO) + 0.437\ (RFC)$	Aplikacija
S19	$F_i = \frac{1}{E_i} = \frac{1}{\sqrt{\frac{\sum_{j=1}^{j=N} E_j}{N}}}$	Razred
S20	Ni enačbe	Razred
S21	$InRank(A) = (1 - d) + d * \left(\frac{InRank(T1)}{C(T1)} + \cdots + \frac{InRank(Tn)}{C(Tn)}\right)$ $OutRank(A) = (1 - d) + d * \left(OutRank(P_1) + \cdots + OutRank(P_n)\right)$ $I(A) = \frac{OutRank(A)}{OutRank(A) + InRank(A)}$	Aplikacija

Iz tabele (glej Tabela 8) je razvidno, da obstaja veliko različnih načinov za napovedovanje vzdrževalnosti. V naslednjem delu naloge bomo nekatere tudi natančneje opisali.

2.5.3 Uporabljene metrike

V tem poglavju smo zapisali, katere študije so uporabile katere izmed metrik za napovedovanje vzdrževalnosti oziroma so ugotavljale povezavo med vzdrževalnostjo in metrikami določenega tipa. Tako smo odgovorili na raziskovalno vprašanje 2.3 – Katere metrike in kateri tipi metrik so uporabljeni za napovedovanje vzdrževalnosti. Metrike smo razdeli na 4 tipe:

- Tradicionalne metrike: metrike izvorne kode, ki niso omejene na objektno orientirane programske jezike. Primer takšnih metrik so recimo LOC, Halstead kompleksnost in število sprememb izvorne kode.
- Objektno-orientirane metrike: metrike izvorne kode, ki prikazujejo dodatne informacije v objektih programskih jezikih in povezavah med razredi.
- Načrtovalske metrike: so uporabljene v času snovanja programov, za njihovo izmero ne potrebujemo izvorne kode. Primer so število razredov v UML diagramih.
- Procesne metrike: so metrike, ki niso povezane z izvorno kodo ali programom, ampak so odvisne od drugih dejavnikov. Primer je recimo strošek spremembe.

V tabeli (glej Tabela 9) lahko vidimo, katere metrike so bile uporabljene v določenih člankih.

Tabela 9: Uporabljeni tipi metrik pri študijah.

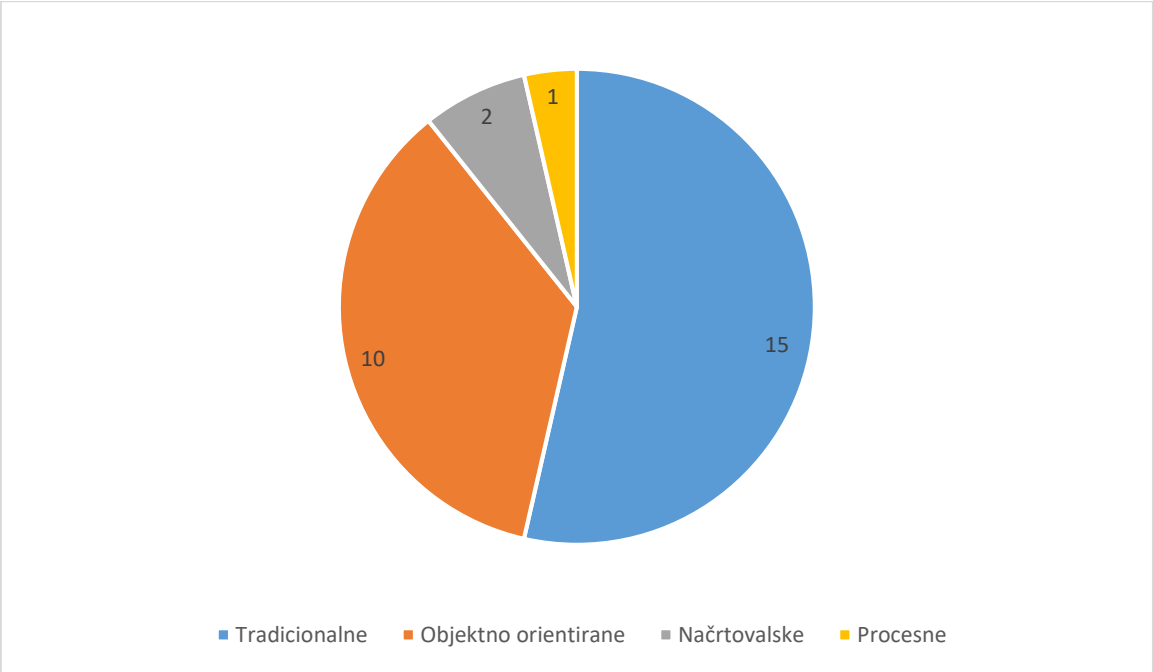
Študija\Tip	Tradicionalne	Objektno orientirane	Načrtovalske	Procesne
S1	x			
S2	x			
S3		x	x	
S4			x	
S5	x	x		
S6	x			
S7	x	x		
S8		x		
S9	x			
S10				
S11		x		
S12	x	x		
S13	x			x
S14	x			
S15	x	x		
S16	x			
S17	x			
S18		x		
S19	x	x		
S20	x			
S21	x	x		

Natančnejšo tabelo, katera metrika je bila uporabljena, prikazuje tabela 9. Pri analizi smo si pomagali s številom, kolikokrat je bil uporabljen vsak izmed tipov metrik. Kot lahko vidimo iz prejšnje tabele, nekatere izmed študij uporabijo tako tradicionalne kot objektno orientirane metrike, zato je število metrik višje od števila študij.

Tabela 10: Število študij, ki uporablja določen tip metrik.

Tip	Število študij
Tradicionalne	15
Objektno-orientirane	10
Načrtovalske	2
Procesne	1

S pomočjo te tabele (glej Tabela 9) lahko izrišemo graf (glej Slika 4) iz katerega je lepše razvidno, kateri so uporabljeni tipi metrik.



Slika 4: Število študij, ki uporabljajo določen tip metrik.

Iz podatkov je jasno razvidno, da največ primarnih študij uporablja za napovedovanje težavnosti vzdrževanja tradicionalne metrike, in sicer 15 študij od skupno 21 izbranih študij. Pogosto uporabljen tip metrik za napovedovanje vzdrževalnosti so tudi objektno-orientirane. V tej skupini predvsem izstopajo tiste študije, ki napovedujejo vzdrževalnost z uporabo umetne inteligence. Jasno je razvidno, da se metrike v vseh teh študijah ponavljajo in tako večinoma metrike v tej skupini uporabljajo umetno inteligenco.
Od ostalih nekoliko izstopata S3 in S4, ki napovedujeta vzdrževalnost na podlagi UML diagramov, še preden sistem implementiramo oz. je dostopna izvorna koda, ter uporabljata načrtovalske metrike, in S13, ki pri napovedi uporabi strošek dela, ki ga predstavlja določena sprememba ter tako spada pod procesne metrike.

2.5.4 Množica podatkov

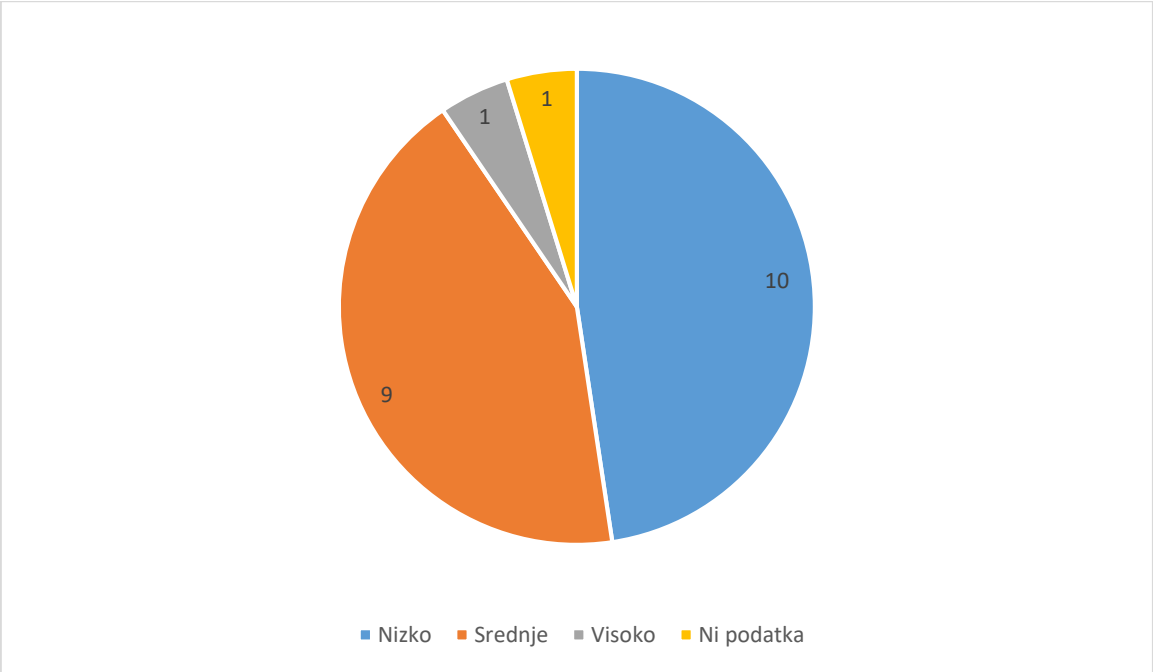
Zapisali smo, kateri projekti so bili uporabljeni pri analizi vzdrževalnosti, kateri programski jezik in tip programskega jezika je bil uporabljen, na kolikem številu projektov je bila določena metrika testirana ter kako veliki so bili ti projekti. Vrednosti smo določili glede na vrednosti v formi za ekstrakcijo podatkov, natančnejše podatke lahko najdemo v tabeli 11. V tem poglavju smo odgovorili na raziskovalna vprašanja 3, 3.1 in 3.2.

Tabela 11: Uporabljeni projekti

ID	Programski jezik	Število	Velikost	Tip projektov	Koda
S1	Proceduralni (C)	Nizko	Veliki	Ni podatka	Ni dostopna
S2	Proceduralni (FORTRAN in C++)	Srednje	Različno	Različno	Ni dostopna
S3	Objektni (Ni podatka)	Srednje	Ni podatka	Ni podatka	Ni dostopna
S4	Modeling (UML)	Srednje	Srednji	Ni podatka	Ni dostopna
S5	Objektni (ADA)	Nizko	Majhni	Administracijski sistemi	Ni dostopna
S6	Ni podatka	Ni podatka	Ni podatka	Ni podatka	Ni dostopna
S7	Objektni (ADA)	Nizko	Majhni	Administracijski sistemi	Ni dostopna
S8	Objektni (Java)	Visoko	Različno	Različno	Dostopno
S9	Objektni (Ni podatka)	Nizko	Ni podatka	Administracijski sistemi	Ni dostopna
S10	Ni podatka	Nizko	Ni podatka	GUI aplikacije	Ni dostopna
S11	Objektni (Ni podatka)	Nizko	Srednji	Ni podatka	Ni dostopno
S12	Objektni (Java)	Srednje	Srednji	GUI aplikacije	Dostopno
S13	Objektni (Java)	Srednje	Veliki	Ni podatka	Delno dostopna
S14	Objektni (Java)	Nizko	Veliki	Administracijski sistemi	Ni dostopno
S15	Objektni (Java)	Srednje	Veliki	GUI aplikacije	Dostopno
S16	Objektni (C++)	Srednje	Majhni	Knjižnice	Ni dostopna
S17	Proceduralni (C in Assembly)	Nizko	Veliki	GUI aplikacije	Delno dostopna
S18	Objektni (Java)	Srednje	Veliki	Različno	Dostopna
S19	Objektni (ADA)	Nizko	Majhni	Administracijski sistemi	Ni dostopna
S20	Objektni (ADA)	Nizko	Majhni	Administracijski sistemi	Ni dostopna
S21	Objektni (Java)	Srednje	Različno	Knjižnice	Dostopna

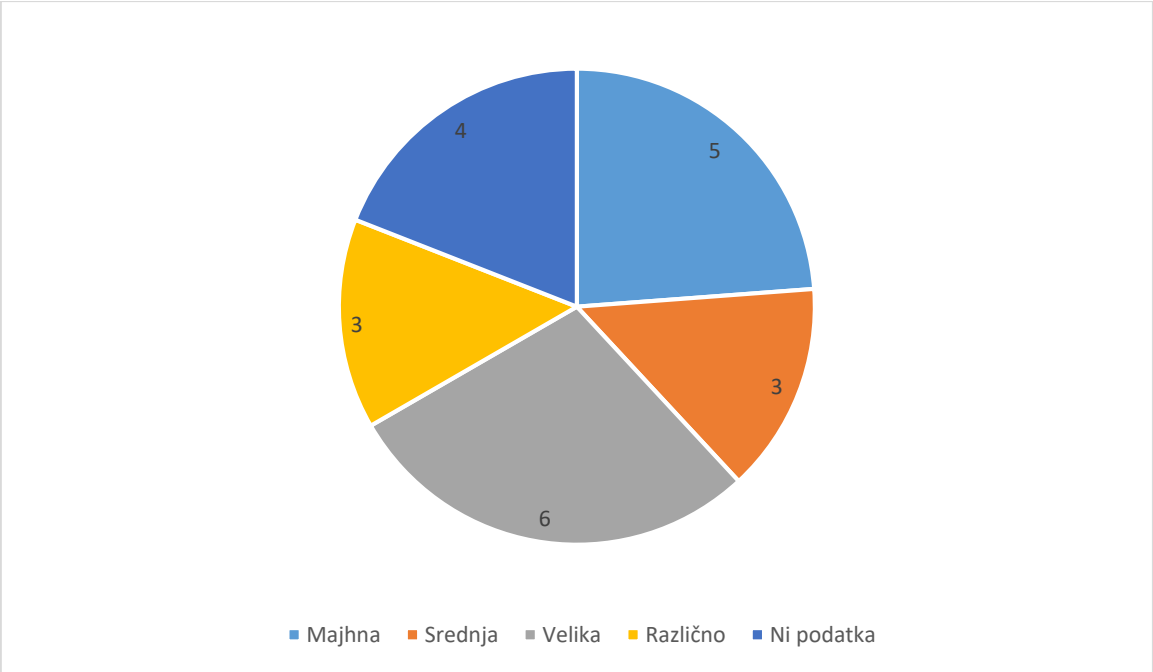
V nadaljevanju smo analizirali tabelo (glej Tabela 10), in sicer tako, da smo podatke prikazali v obliki grafikonov ter jih analizirali. Večina študij je uporabila objektni jezik, saj le tri študije uporabljajo proceduralnega in le ena študija jezik UML. Prav tako so te študije po starosti najstarejše. Edina izjema je študija S17, ki analizira spremembe na operacijskem jeziku Linux, ki je napisan v C in tako spada pod proceduralne jezike. Iz tega lahko ugotovimo, da je za metrike pomembno tudi to, da jih lahko uporabimo pri objektno-orientiranih jezikih in tako dobimo podatke o vzdrževalnosti le-teh.

Kot naslednje smo analizirali število projektov. Kot je razvidno iz grafikona, je v večini uporabljeno nizko število projektov, kar pomeni, da so pri validaciji uporabili le enega ali dva različna projekta. Pri tem predvsem izstopajo študije, ki uporabljajo umetno inteligenco, saj vse izmed njih uporabijo ista dva projekta – QUES in UIMS. Tako vsi projekti, ki pri napovedovanju vzdrževalnosti uporabijo umetno inteligenco, padejo v to skupino.



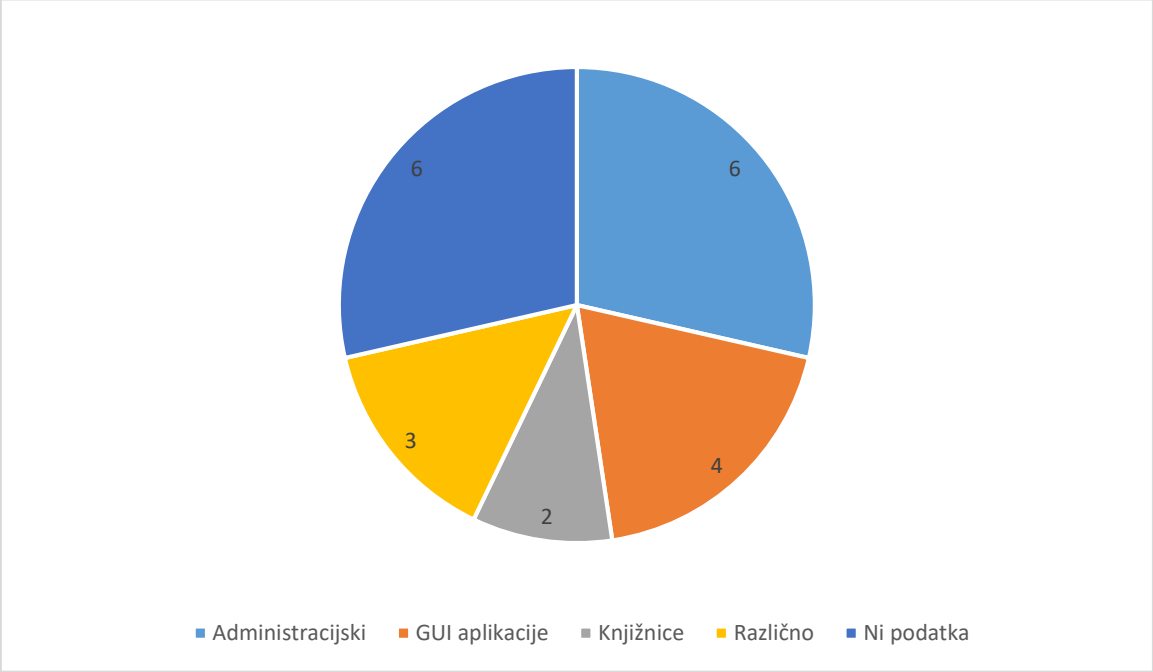
Slika 5: Število projektov.

Prav tako je iz grafikona razvidno (glej Slika 5), da je dokaj visoko število tudi študij, ki uporabijo srednje število projektov, kar pomeni, da so uporabili od 3 do 10 projektov. Zanimivo je, da je le ena študija uporabila visoko število projektov z izvedeno validacijo na več kot desetih projektih. Za prenos določenih metrik in modelov v množično uporabo bi verjetno bilo potrebno vsako metriko validirati na več projektih, saj kot bo razvidno iz tipa programske opreme, je bilo veliko metrik preizkušenih le na določeni domeni. V naslednjem delu smo analizirali velikost projektov, ki so bili uporabljeni.



Slika 6: Velikost projektov.

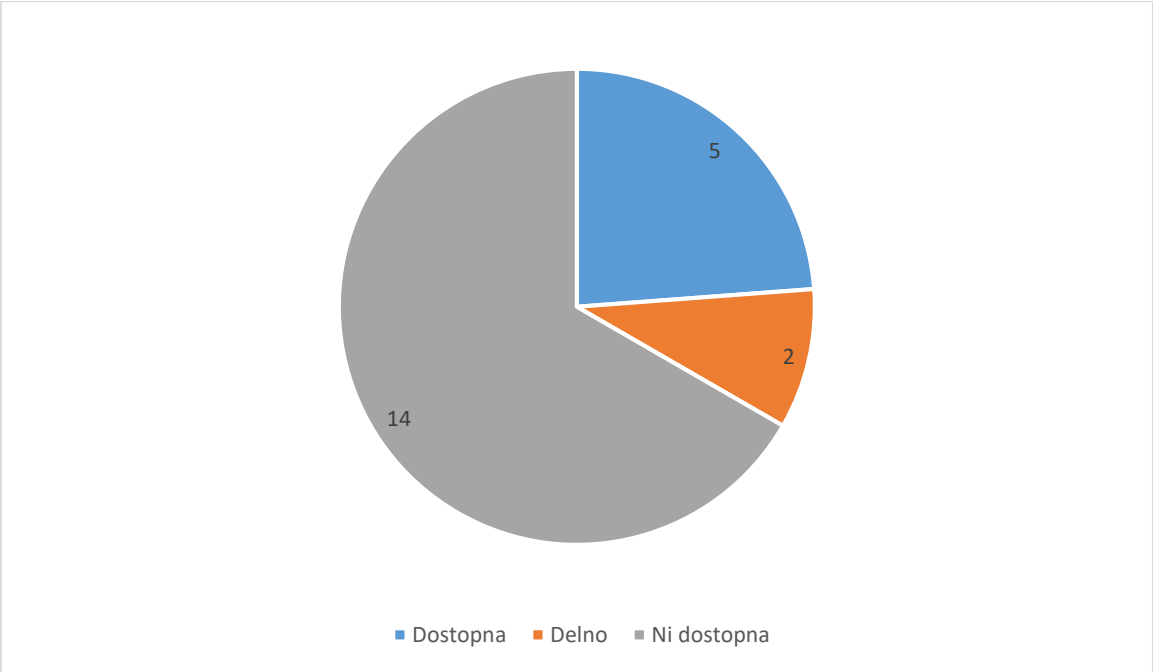
Kot lahko vidimo, je tortni grafikon (glej Slika 6) glede velikosti projektov dokaj podobno razdeljen med vse sklope. Ne moremo izpostaviti ničesar posebnega, razen tega, da je presenetljivo visoko število projektov, kjer je bila velikost označena kot majhna. Ti projekti so obsega pod 500 vrstic kode. Variabilnost metrik za napovedovanje vzdrževalnosti pri takšni velikosti je dokaj spremenljiva, zato menimo, da bi morale vse študije izračunati vzdrževalnost za vsaj en projekt srednje velikosti. V naslednjem razdelku smo analizirali tip projektov (glej Slika 7).



Slika 7: Tip projektov.

Kot je razvidno, sta le dve študiji uporabili projekte iz različnih domen. Vsekakor bi za splošno validacijo določene metrike ali metodologije bilo treba opraviti validacijo na več področjih in tako zagotoviti, da bo metrika delovala tudi na preostalih področjih. Razlog je v tem, da se izvorna koda projektov spreminja glede na domeno, pod katero spada določen projekt.

Kot je razvidno iz naslednjega grafikona (glej Slika 8), večina projektov, ki so jih avtorji študij uporabili, ne pove dovolj o projektih, da bi lahko pridobili njihovo izvorno kodo ali so uporabili lastniško programsko opremo ter tako izvorna koda programov v večini študij ni dostopna. Posledično je zato težko ponoviti katerega izmed eksperimentov avtorjev v njihovih študijah. Kljub temu je bilo med 21 študijami 7 takšnih, kjer je izvorna koda projektov v celoti oziroma vsaj delno dostopna. Nekatere izmed teh odprtokodnih programov smo uporabili v lastnem eksperimentalnem delu, saj smo pričakovali, da bodo na njih izbrane metrike iz teh študij dosegle želen rezultat, kot so ga dosegli avtorji študij.



Slika 8: Dostopnost izvorne kode projektov.

2.5.5 Uporabljena orodja

V tem poglavju bomo zapisali, katere študije so uporabile določeno metrično orodje in ali je to orodje dostopno. Pri tem smo odgovorili na raziskovalni vprašanji 4 in 4.1. V tabelo (glej Tabela 12) smo vključili vse študije, ki so podale informacije, katero orodje je bilo uporabljeno za izračun vzdrževalnosti.

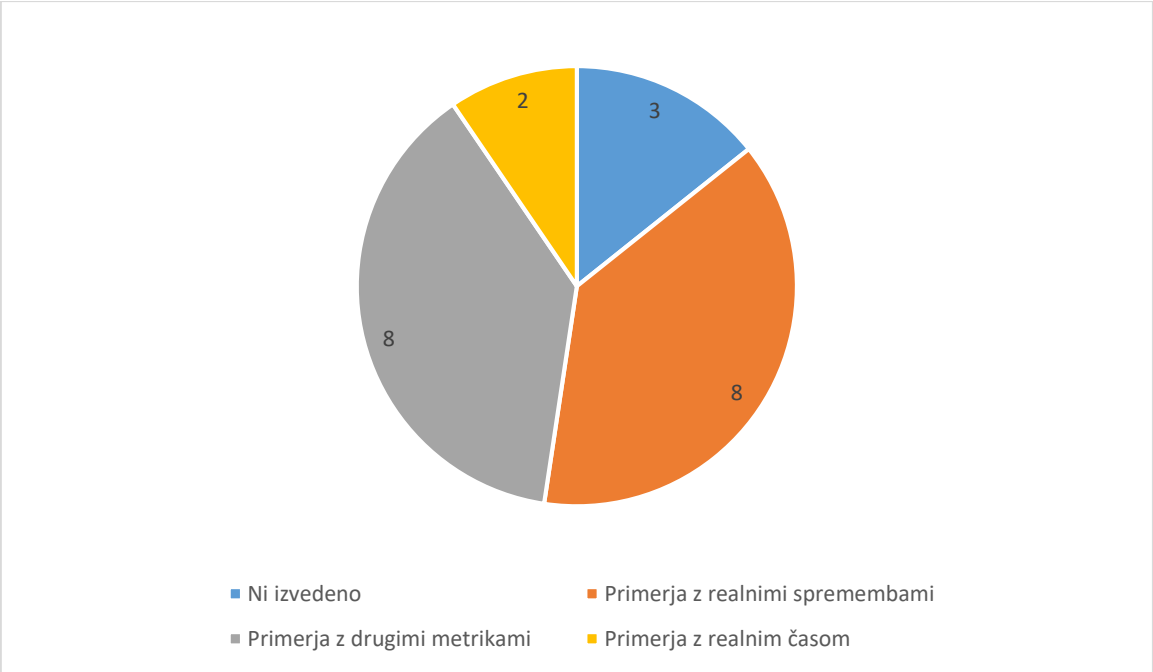
Tabela 12: Uporabljena orodja v študijah.

Članek	Ime orodja	Prosto dostopno
S9	Visual Studio 2008 + “Metrics Analyzer” module	Ni dostopno
S12	JHawk	Delno
S14	Ckjm	Da
S15	MMIS	Ni dostopno
S17	srcSlice CodeSurfer	Dostopno Ni dostopno
S18	Ckjm jDepend	Dostopno Ni dostopno
S20	Anfis Tool kit in MATLAB	Delno
S21	ObjectAid UML Explorer Jhawk JDepend Classycle	Dostopno Delno Dostopno Dostopno

Kot lahko vidimo, je število študij v tabeli le 8 raziskav, kar pomeni, da le 8 od 21 študij pove, katero orodje so uporabili za pridobitev rezultatov, čeprav so verjetno vse študije uporabile določeno orodje.Orodja se med študijami razlikujejo in so bila uporabljena glede na to, kar je študija potrebovala. Iz tabele je razvidno, da ima večina orodij na voljo brezplačno preizkusno verzijo, medtem ko je celotna verzija programov plačljiva. Izjema so odprtokodne rešitve, ki so bile uporabljene v nekaterih študijah – Ckjm v S14, srcSlice v S17 ter JDepend in Classycle v S21. Iz napisanih orodij smo za nadaljnjo raziskavo izbrali tista, ki omogočajo pridobivanje vrednosti o metrikah iz javanske izvorne kode. To so JHawk, JDepend, Classycle in Ckjm.

2.5.6 Pridobljeni rezultati

V tem poglavju smo predstavili, kako so avtorji primerjali rezultate ter validirali svojo metodo ter kakšne so njihove ugotovitve. V tem poglavju smo odgovorili na raziskovalno vprašanje 5, 5.1 in 5.2.



Slika 9: Analiza rezultatov.

Kot lahko vidimo na sliki (glej Slika 9), so avtorji pravilnost dobljenih rezultatov validirali na različne načine. Kot smo že zapisali v opisu ekstrakcijske forme, smo vsak članek uvrstili v eno izmed kategorij in tako prišli do ugotovitve, da se najpogosteje uporablja primerjava z realnim številom sprememb oziroma primerjava z eno izmed že obstoječih in validiranih metrik. To pomeni, da so avtorji člankov v večini primerov spremljali razvoj določene programske opreme, opazovali število sprememb in na podlagi tega validirali svojo metriko oziroma pri tem uporabili gibanje že validirane metrike in rezultate le te primerjali s svojo metriko.

Kot je razvidno iz priloge D, pri primerjavi z že obstoječimi metrikami prevladuje metrika MI, saj je prva, ki je obstajala na tem področju in je tudi že validirana, saj prikazuje realno vrednost, kako težavno bo vzdrževanje določene programske opreme. Na podlagi tega smo se tudi odločili, da smo indeks vzdrževalnosti izbrali kot osnovo, s pomočjo katere smo analizirali pravilnost ostalih metrik.

Odgovor na raziskovalno vprašanje 5.2 je različen med članki ter ni mogoče sprejeti neke skupne ugotovitve. Večina avtorjev na koncu člankov trdi, da je našla metriko, ki jo lahko uporabimo za napovedovanje vzdrževalnosti in je na tem področju možna nadaljnja raziskava.

2.6 Izbira metrik in modelov za vzdrževalnost

V tem poglavju smo zapisali razlog za neizbiro metrik posameznegačlanka. V spodnji tabeli (glej Tabela 13) lahko vidimo razlog, zakaj se nismo odločili za določeno metriko.

Tabela 13: Izbira metrik.

Študija	Izbrano	Razlog
S1	Da	
S2	Da	
S3	Ne	Indeks ni dovolj natančno definiran, da bi ga lahko uporabili.
S4	Ne	Uporabljene metrike je možno izračunati le v UML.
S5	Ne	Uporaba umetne inteligence.
S6	Ne	Formula metrike ni javno podana.
S7	Ne	Uporaba umetne inteligence.
S8	Ne	Uporaba umetne inteligence.
S9	Ne	Pridobivanje vrednosti metrik ni mogoče.
S10	Ne	Metrika ni dovolj izpeljana, da bi jo lahko uporabili.
S11	Ne	Formula indeksa ni točno določena.
S12	Ne	Pridobivanje vrednosti metrik ni mogoče.
S13	Ne	Pridobivanje vrednosti metrik ni mogoče.
S14	Ne	Formula indeksa ni točno določena.
S15	Ne	Formula indeksa ni točno določena.
S16	Da	
S17	Ne	Formula indeksa ni točno določena.
S18	Da	
S19	Ne	Uporaba umetne inteligence.
S20	Ne	Uporaba umetne inteligence.
S21	Ne	Metrika ni dovolj izpeljana, da bi jo lahko uporabili.

Kot je razvidno iz tabele, smo se odločili uporabiti metrike iz štirih različnih člankov, in sicer tri različne oblike formule za indeks vzdrževalnosti, kjer se nekoliko spreminjajo le formule za izračun. Prav tako smo v eksperimentalnem delu uporabili formuli iz članka S18, ki uporabljata Martinovo oziroma CK zbirko metrik. Te lahko uporabimo pri izračunih in je tako tudi uporabljena formula nekolika drugačna od tiste, ki je uporabljena pri indeksu vzdrževalnosti.

V večini primerov je razlog neizbire določenega članka ta, da je ali ta na področju uporabe umetne inteligence, ki ima drugačen način izračuna od želenega, ali pa formula indeksa oziroma kako priti do vrednosti metrike ni dovolj natančno določena.

2.7 Opis izbranih metrik

Za nadaljnjo analizo smo izbrali metrike, ki smo jih lahko pridobili z določenimi orodji ter so dovolj natančno opisane v člankih, da jih lahko izračunamo.

2.7.1 Osnovni indeks vzdrževalnosti (S1)

Indeks vzdrževalnosti (Maintainability index) je prvotna in najstarejša metrika, ki se lahko uporabi za napoved vzdrževalnosti. Je tudi ena izmed najbolj pogostih metrik za napovedovanje vzdrževalnosti. V članku S1 je opisana ena izmed osnovnih formul (glej Enačba 2.15) za napovedovanje vzdrževalnosti.

$$MI = 171 - 3.42 * \ln(aveE) - 0.23 * aveVG2 - 16 * (aveLOC) \quad (2.15)$$

V formuli predstavlja *aveE* povprečno težavnost, *aveVG2* povprečno ciklomatično kompleksnost in *aveLOC* povprečno število vrstic kode.

2.7.2 Izboljšan indeks vzdrževalnosti (S2)

Članek predstavi izboljšano verzijo indeksa (Improved Maintainability index), predstavljenega v S1. Spremenjen je drugi parameter pri Halstead Volume, ki je bil iz 3.42 povečan na 5.2, vendar se je tudi spremenila definicija parametra, tako da je končni rezultat zelo podoben prejšnjemu. Formula (glej Enačba 2.16) in metrike so v vseh ostalih pogledih enake kot pri S1 ter je to torej le manjša nadgradnja indeksa.

$$Maintainability\ Index = 171 - 5.2 * \ln(aveV) - 0.23 * aveVG2 - 16 * (aveLOC) \quad (2.16)$$

V formuli predstavlja *aveE* povprečno težavnost, *aveVG2* povprečno ciklomatično kompleksnost in *aveLOC* povprečno število vrstic kode. Glede na prejšnjo verzijo indeksa se je nekoliko spremenila vrednost parametrov.

2.7.3 Indeks vzdrževalnosti na osnovi LOC (S16)

Članek predstavi novo metriko za izračunanje vzdrževalnosti, ki prav tako temelji na osnovnem indeksu vzdrževalnosti. Pri tem formula (glej Enačba 2.17) ostane enaka kot pri osnovnih indeksih, razlika je le ta, kako se izračuna vrednost spremenljivk. Pri tem se izračuna vrednost le z uporabo metrike *LOC*.

$$MI = 171 - 5.2 * \ln(HV) - 0.23 * CC - 16.2 * \ln(LOC) \tag{2.17}$$

Pri tem se *HV* (Halstead volumen) in *CC* izračunata iz spodnjih formul (glej Enačba 2.18, 2.19).

$$HV = 45 * LOC - 428 \tag{2.18}$$

$$CC = 0.22 * LOC + 1.9 \tag{2.19}$$

Za izračun indeksa vzdrževalnost tako potrebujemo le povprečno število vrstic kode modelov v izvorni kodi.

2.7.4 Indeks vzdrževalnosti na osnovi nabora Martin metrik (S18)

Nabor Martin metrik (Martin suite) je skupina metrik, ki so bile predstavljene leta 2003 in so uporabljene na ravni paketa. Metrike v tej skupini so povezane predvsem s povezavami med razredi, njihovo uporabo in kompleksnostjo uporabe.

Pri razvoju metrike je avtor uporabil linearno regresijo z vsemi možnimi spremenljivkami, pri čemer je ugotovil, da imajo določene metrike znotraj skupine prenizke intervale zaupanja, da bi jih lahko uporabili za izračun vzdrževalnosti programske opreme. Tako je iz preostalih štirih metrik in linearno regresijo pridobil sledečo formulo (glej Enačba 2.20).

$$MI = 55.855 - 13.678\ (D) - 2.577\ (Ca) - 0.154\ (CC) + 0.938\ (Ce) \tag{2.20}$$

V formuli 2.20 predstavlja *D* razdaljo paketa od idealnega izraza, *Ca* predstavlja število razredov v drugih paketih, ki so odvisni od razreda, *Ce* število razredov v drugih paketih, od katerih je odvisen razred, in *CC* predstavlja število razredov.

2.7.5 Indeks vzdrževalnosti na osnovi nabora CK metrik (S18)

Metrike v naboru oziroma Chidamber in Kemerer metrike so nastale leta 1993 in so ene najbolj pomembnih objektno orientiranih metrik. V to skupino spadajo metrike, kot so globina drevesa dedovanja in število metod v določenem razredu.

Avtor je pri razvoju formule za izračun indeksa vzdrževalnosti uporabil enako tehniko, kot je bila opisana v prejšnjem podpoglavju, kjer je v tem primeru interval zaupanja bil dovolj velik le pri dveh metrikah iz skupine. Tako je prišel do končne formule (glej Enačba 2.21).

$$MI = 42.007 - 1.181\ (CBO) + 0.437\ (RFC) \tag{2.21}$$

V formuli *CBO* (Coupling between Objects) predstavlja število razredov, ki so vezani na določen razred, spremenljivka *RFC* (Reponse for a Class) pa pomeni število različnih metod, ki jih lahko izvede določen razred, ko dobi sporočilo.

3 PREGLED ORODIJ ZA IZRAČUN METRIK ZA NAPOVEDOVANJE VZDRŽEVALNOSTI

V tem delu magistrske naloge smo pregledali obstoječa orodja, ki smo jih zasledili med sistematičnim pregledom literature, in tista, ki smo jih našli na spletu in bi lahko ustrezala našim potrebam za empirični del raziskave. Prav tako smo izbrali orodja iz članka [31], kjer je bil izveden sistematični pregled literature orodij za objektno-orientirane metrike. Iz pregleda smo se osredotočili na tista orodja, ki omogočajo pridobivanje vrednosti želenih metrik. Cilj je bil izbrati tisto orodje, ki je pomagalo pri pridobitvi rezultatov za napovedovanje vzdrževalnosti, saj smo nameravali s tem orodjem izvesti praktični del magistrske naloge. Če ni na voljo orodja, ki bi omogočalo zajemanje vseh potrebnih metrik, smo poizkušali najti takšno kombinacijo orodij, kjer smo lahko uporabili čim manj orodij za izračun vzdrževalnosti. V tem poglavju smo preizkusili in analizirali deset orodij, s katerimi lahko pridobimo določene metrike. Za izračun vseh petih indeksov vzdrževalnosti namreč potrebujemo vrednosti metrik ciklomatična kompleksnost, LOC, Halsteadov trud in obseg, D, Ca, Ce, CBO in RFC.

Pri izbiri orodja smo si pomagali z določenimi kriteriji, ki so:

- orodje mora podpirati analizo izvorne kode v Javi,
- orodje mora biti brezplačno oziroma mora imeti brezplačno testno verzijo,
- orodje mora omogočati pridobivanje določenega podatka, ki ga ostala orodja na tem seznamu ne podpirajo in
- orodje mora imeti sposobnosti analizirati celoten večji projekt z enim pogonom programa.

Izbrali bomo tisto orodje, ki bo v največji meri zadostilo našim kriterijem. Po opisu orodij sledi tabela, v kateri je označeno, za katera orodja smo se odločili in za katera ne ter podali razloge za našo odločitev.

3.1 Orodje JHawk

JHawk je orodje za statično analizo kode. Za svoje delovanje potrebuje izvorno kodo projekta, s pomočjo katere izračuna razne metrike in karakteristike kode. Metrike, ki jih ponuja, so recimo metrika obsega, kompleksnosti, relacij med razredi in paketi ter relacija med samimi razredi in med paketi. Orodje ima na voljo grafičen vmesnik, ki omogoča prikazovanje rezultatov. Orodje je plačljivo za uporabo, ima pa za uporabo na voljo preizkusno verzijo, kjer je omogočeno analiziranje največ petih razredov, da lahko uporabnik vidi, katere metrike orodje izračuna. Orodje je dostopno na http://www.virtualmachinery.com/jhawkfilebased.htm.

3.2 Orodje Sonar

SonarQube je odprto-kodna platforma za nadzor kakovosti kode, ki je namenjena neprestani analizi kode ter izračunanju tehničnih lastnosti kakovosti kode na ravni projekta, prav tako pa na manjših komponentah, vse do nivoja metode. Orodje omogoča samodejno odkrivanje in gibanje ocen skozi čas glede beleženja napak ter hroščev v kodi, pomanjkljivosti in ostalih tehničnih lastnosti projekta. Orodje ima grafičen spletni vmesnik, kjer lahko te rezultate pregledamo. Za delovanje potrebujemo orodje, ki pregleda kodo in se imenuje Sonar Scanner. Scanner deluje v ukazni vrstici in ga prožimo z uporabo .bat datoteke scannerja, ko se v ukazni vrstici nahajamo v domačem direktoriju projekta, ki ga želimo analizirati.

Orodje izračuna različne karakteristike kode, vendar so se v novejših verzijah odločili povečati pozornost k uporabi kompleksnejših metrik za napovedovanje težavnosti vzdrževanja kode, kot je metodologija Sqale, vendar so zato prenehali podpirati osnovne metrike strukture kode. V starejših verzijah je orodje omogočalo izračun tako metrik kompleksnosti kakor tudi povezav med razredi z uporabo raznih dodatkov, vendar ti trenutno niso več podprti. Orodje je dostopno na http://www.sonarqube.org/.

3.3 Orodje JDepend

Orodje JDepend za svoje delovanje potrebuje izvorno kodo projekta, ki jo pregleda in določi metrike na ravni paketa. JDepend dovoljuje merjenje kakovosti strukture kode in povezav med paketi ter stopnjo ponovne uporabe in odvisnosti med njimi. Orodje omogoča več načinov uporabe, najprimernejši za nas je z uporabo paketa textui, ki zapiše vse izračunane metrike v datoteko.

```
java jdepend.textui.JDepend  »Pot do projekta«
```

Orodje je odprtokodno ter tako brezplačno za uporabo in je dostopno na http://clarkware.com/software/JDepend.html.

3.4 Orodje Ckjm

Orodje izračuna objektno-orientirane metrike, ki sta jih predlagala Chidamber in Kemerer [32]. Program procesira prevedene javanske datoteke, kjer je bitna koda programa in za vsak razred izračuna naslednjih šest metrik, ki sta jih prej omenjena avtorja predlagala, in sicer WMC, DIT, NOC, CBO, RFC in LCOM. Razen tega lahko pridobimo še podatke o Ca in NPM. Program se izvede iz ukazne vrstice, kjer je treba uporabiti sledeči ukaz.

```
java -jar »Pot do orodja«\ckjm-1.9.jar »Pot do projekta«\*.class
```

Program poženemo tako, da poženemo jar datoteko in nato dodamo poljubno število poti do datotek, za katere želimo, da orodje izračuna vse metrike. Težava programa je v tem, da lahko tako analiziramo vse .class datoteke na neki poti, vendar program ni sposoben

preveriti, če obstajajo paketi znotraj teh datotek, tako da bi za vsak paket v projektu morali dodati nov argument v ukazno vrstico.
Program je dostopen na: http://www.spinellis.gr/sw/ckjm/.

3.5 Orodje Classycle

Classycle je orodje za analizo izvorne kode in povezav med paketi v javanskih aplikacijah ali knjižnicah. Primerno je za pridobivanje podatkov o cikličnih odvisnostih med razredi in paketi. Classycle je dokaj podobno orodju JDepend, le da orodje JDepend omogoča tudi analizo odvisnosti na nivoju paketov, medtem ko se orodje Classycle osredotoča na odvisnosti med razredi.
Classycle tako omogoča pregledovanje izvorne kode za neželene razrede v primerjavi s tem, kako smo jo sami definirali v izbrani datoteki. S tem nam uspe pregledovati, ali arhitektura naše aplikacije ustreza našim zahtevam ali ne.
Analizo poženemo z uporabo .jar datoteke in navedbo vseh .class datotek programa, ki ga želimo analizirati.

```
java -jar »pot do orodja«\classycle.jar »binarne datoteke«
```

Orodje je odprtokodno in je na voljo na http://classycle.sourceforge.net/.

3.6 Orodje Teamscale

Teamscale je orodje, ki uporabniku analizira kakovost izvorne kode projektov. Namenjeno je beleženju podatkov skozi daljše časovno obdobje ter kako se je koda v tem času spreminjala. Prav tako omogoča avtomatsko odkrivanje napak in pomanjkljivosti v kodi, ki jih je lahko spregledati. Namenjeno je predvsem uporabi pri večjih projektih, kjer se z uporabo orodja izognemo večjim stroškov v prihodnosti. Orodje tako ne prikazuje osnovnih metrik, ki bi jih potrebovali za izračun vzdrževalnosti. Analizo poženemo z uporabo spletnega uporabniškega vmesnika, kjer so tudi prikazani rezultati analize.
Orodje je plačljivo, vendar ima na voljo brezplačno preizkusno verzijo. Spletna stran orodja je https://www.cqse.eu/en/products/teamscale/landing/.

3.7 JavaNCSS

JavaNCSS je preprosto orodje, ki se uporabi v ukazni vrstici ter prikaže vrednosti določenih metrik, tako za celoten projekt kot za posamezne razrede in funkcije. Podprte so naslednje metrike:

- NCSS,
- McCabe metrik (CC),
- število metod, razredov, paketov in
- število Javadoc komentarjev v razredu.

Orodje je brezplačno in je na voljo na https://github.com/codehaus/javancss.

3.8 Orodje CLOC

Orodje CLOC prešteje vrstice kode, prazne vrstice ter vrstice, ki vsebujejo komentarje v različnih programerskih jezikih, med katerimi je tudi Java. Če podamo dve različni verziji izvorne kode programa, je orodje sposobno izračunati razliko v vrednosti metrik med verzijami. Orodje je napisano v jeziku Perl.

Orodje poženemo z uporabo spodnje vrstice, ko smo v direktoriju projekta, ki ga želimo analizirati.

```
»pot do orodja«/cloc-»verzija« *.java
```

Orodje je dostopno kot odprtokoden projekt in je posledično brezplačno za uporabo. Orodje je dostopno na https://github.com/AlDanial/cloc.

3.9 Orodje CCCC - C and C++ Code Counter

CCCC je orodje, ki analizira C++ in javanske datoteke ter generira poročilo o različnih metrikah v kodi. Podprte metrike vključujejo:

- LOC,
- McCabe CC in
- metrike CK (razen RFC).

Program poženemo iz projektnega direktorija z uporabo spodnje ukazne vrstice.

```
»pot do orodja«/cccc *.java
```

Orodje je nastalo kot odprtokodni projekt in je posledično brezplačno za uporabo. Orodje je dostopno na https://sourceforge.net/projects/cccc/.

3.10 Orodje CMTJava

CMTJava oziroma orodje za merjenje kompleksnosti (angl. Complexity Measures Tool) v Javi, ki iz izvorne kode projekta izračuna določene metrike kompleksnosti projekta, kot sta CBO in RFC. Orodje poženemo z uporabo ukazne vrstice ter poganjanjem analize programa s spodnjim ukazom.

```
dir *.java /b /s | »pot do orodja«/CMTJava
```

Orodje je plačljivo. Spletna stran orodja je dosegljiva na spletnem naslovu http://www.verifysoft.com/en_cmtx.html.

3.11 Rezultati vrednotenja in izbrano orodje

V tem poglavju smo zapisali, zakaj smo se odločili za uporabo določenega orodja oziroma zakaj smo določena orodja zavrgli. Prav tako smo zapisali, katere metrike smo uporabili pri vsakem izmed naštetih orodij.

Analizirali smo skupno 10 orodij, ki smo jih našli v pregledih literature in so podpirala metrike, ki jih potrebujemo. Pregledali smo rezultate in kako delujejo vsa orodja ter se odločili, katera orodja uporabiti.

Tabela 14: Izbira orodij.

Orodje	Izbrano/Neizbrano, razlog
JHawk	Neizbrano, orodje je plačljivo.
Sonar	Neizbrano, ne ponuja želenih metrik.
JDepend	Izbrano.
Ckjm	Izbrano.
Classycle	Neizbrano, ne ponuja želenih metrik.
Teamscale	Neizbrano, namenjeno za beleženje kakovosti kode skozi čas.
JavaNCSS	Neizbrano, ne ponuja želenih metrik.
CLOC	Neizbrano, ne ponuja želenih metrik.
CCCC – C and C+ Code Counter	Neizbrano, podprte metrike lahko izračunamo s programi, ki imajo več potrebnih metrik.
CMTJava	Izbrano.

V tabeli (glej Tabela 14) lahko vidimo, da večine orodij nismo izbrali, ker ne podpirajo potrebnih metrik oziroma smo za izračun le-teh naleteli na orodje, ki hkrati omogoča izračun več potrebnih metrik. Določena orodja smo izpustili zaradi tega, ker niso brezplačna in smo uspeli najti orodja, ki smo jih lahko uporabili brez plačila.

Kot je razvidno iz tabele, smo za uporabo pri praktičnem delu magistrske naloge izbrali 3 orodja in sicer:

- CMTJava,
- JDepend in
- Ckjm.

S temi tremi orodji pridobimo vse potrebne interne metrike kakovosti kode, ki se uporabijo pri izračunu vzdrževalnosti in smo jih pri sistematičnem pregledu literature izbrali za uporabo v praktičnem delu magistrske naloge. V tabeli (glej Tabela 15) lahko vidimo, katere metrike smo pridobili iz katerega orodja.

Tabela 15: Izbrana orodja in metrike.

Izbrano orodje	CMTJava	JDepend	Ckjm
Halstead Volume	x		
Halsted Effort	x		
CC	x		
LOC	x		
D		x	
Ca		x	
Ce		x	
CBO			x
RFC			x

Orodji CMTJava in JDepend potrebujeta za svoje delovanje le izvorno kodo projekta, medtem ko za uporabo orodja Ckjm potrebujemo prevedeno strojno kodo projekta, tako da smo pri praktičnem delu potrebovali tako izvorno kodo projektov kot tudi prevedeno kodo.

4 IZVEDBA EKSPERIMENTALNEGA DELA

V tem poglavju smo opisali in izvedli eksperimentalni del magistrske naloge.

4.1 Definicija in načrtovanje eksperimenta

V magistrskem delu smo v prejšnjih poglavjih opravili in opisali sistematični pregled literature, kjer smo izbrali metrike, ki jih lahko uporabimo za napovedovanje vzdrževalnosti. Nato smo izbrali orodja, z uporabo katerih smo pridobili potrebne interne metrike za izračun izbranih metrik iz sistematičnega pregleda literature. V naslednjem poglavju je opisan postopek izbire projektov, ki smo jih analizirali.

Z uporabo orodij smo dobili metrične rezultate za vse izbrane projekte. Kot prvo smo uporabili Pearsonovo korelacijo in nato F-test, kjer bo indeks vzdrževalnosti, ki je bil že večkrat validiran, služil kot metrika, glede na katero smo primerjali ostale metrike. Najprej smo analizirali koeficient korelacije med rezultati metrik. Za tem smo analizirali rezultate statističnega testa F-test. Rezultate smo nato združili in predstavili končne ugotovitve. Po opravljeni statistični analizi rezultatov smo opisali, kaj v praktičnem smislu pomenijo naši rezultati in kaj lahko na podlagi rezultatov ugotovimo.

V zadnjem podpoglavju smo zapisali še možna tveganja in grožnje za veljavnost eksperimenta, ki smo jih zaznali.

Kot smo že napisali, smo pri analizi uporabili dva testa, s katerima smo pokazali korelacijo med različnimi metrikami in ustreznost delovanja metrik. To sta Pearsonova korelacija in F-test, ki smo ju natančneje opisali v naslednjih poglavjih.

4.1.1 Pearsonova korelacija

Pearsonova korelacija predstavlja mero linearne korelacije med dvema spremenljivkama X in Y [33]. Vrednost mere je med -1 in 1. Bolj kot se vrednost približuje številki 1, večja je korelacija med spremenljivkama, medtem ko je pri -1 korelacija popolnoma nasprotna. Razvita je bila na začetku 20. stoletja s strani Karla Pearsona in je od tedaj pogosto uporabljena v znanosti za ugotavljanje, ali sta določeni dve spremenljivki korelirani.

Formula za izračun korelacije je sledeča (glej Enačba 4.1):

$$r = \frac{\sum XY - \frac{(\sum x)(\sum y)}{n}}{\sqrt{\left(\sum X^2 - \frac{(\sum X)^2}{n}\right)\left(\sum Y^2 - \frac{\sum Y^2}{n}\right)}} \tag{4.1}$$

V enačbi so x in y točke spremenljivk, n je število točk v vsaki množici. Tako lahko dobimo stopnjo korelacije med dvema neodvisnima spremenljivkama.

V praktičnem delu magistrske naloge smo kot prvo spremenljivko vzeli indeks vzdrževalnosti, za katerega je v člankih dokazano, da predstavlja realno stanje vzdrževalnosti. Kot drugo spremenljivko smo uporabili rezultat vrednosti ene izmed ostalih štirih metrik, ki smo jih izbrali v sistematičnem pregledu literature. Pri tej spremenljivki smo iskali takšno vrednost korelacije, ki se čim bolj približa enemu izmed limitov, saj to pomeni, da lahko iz vrednosti indeksa vzdrževalnosti ugotovimo, kako se giblje vrednost vzdrževalnosti druge metrika in tako preko na podlagi vrednosti indeksa vzdrževalnosti s pomočjo vrednosti določenega indeksa napovemo vzdrževalnost programov.

4.1.2 F-Test primerjava variance med dvema spremenljivkama

F-test je uporabljen za ugotavljanje, ali je varianca med dvema spremenljivkama enaka. Formulo sta leta 1967 razvila Snedecor in Cochran [34]. Test je lahko uporabljen za enostransko ali dvostransko testiranje, kjer je dvostransko testiranje alternativa enostranskemu ter se uporablja za potrjevanje hipoteze, da sta varianci dveh spremenljivk različni. S testom poizkušamo ugotoviti, ali se nova spremenljivka bolj spreminja kot obstoječa oziroma ali je njena vrednost podobno porazdeljena.
Postopek izračuna je naslednji. Najprej izračunamo povprečje (glej Enačba 4.2),

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{4.2}$$

nato izračunamo varianco spremenljivke (glej Enačba 4.3)

$$S_X^2 = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \bar{x})^2 \tag{4.3}$$

in nato vrednost F (glej Enačba 4.4).

$$F = \frac{S_X^2}{S_Y^2} \tag{4.4}$$

Iz vrednosti F lahko ugotovimo, če je hipoteza zavrnjena ali potrjena, glede na to, ali je vrednost dovolj visoka ali nizka. Želena dobljena vrednost je 1, kar bi pomenilo, da je varianca spremenljivk enaka. V našem primeru smo uporabili enostransko testiranje, saj poizkušamo ugotoviti ujemanje varianc med dvema spremenljivkama. Iz rezultata F, ki je blizu vrednosti 1, lahko sklepamo, da je porazdelitev med spremenljivkama podobna, kar bi pomenilo, da imata podobno porazdelitev. Posledično glede na vrednost korelacijskega koeficient lahko z uporabo vrednosti referenčne spremenljivke napovemo mejne vrednosti za ostale spremenljivke. V kolikor vrednost F odstopa od 1 za večjo vrednost, lahko glede na višino povprečne vrednosti spremenljivk potrdimo trditev, da vrednosti izbranega indeksa ne more uporabljati za napovedovanje vzdrževalnosti.

4.2 Opis in postopek izbire odprtokodnih projektov

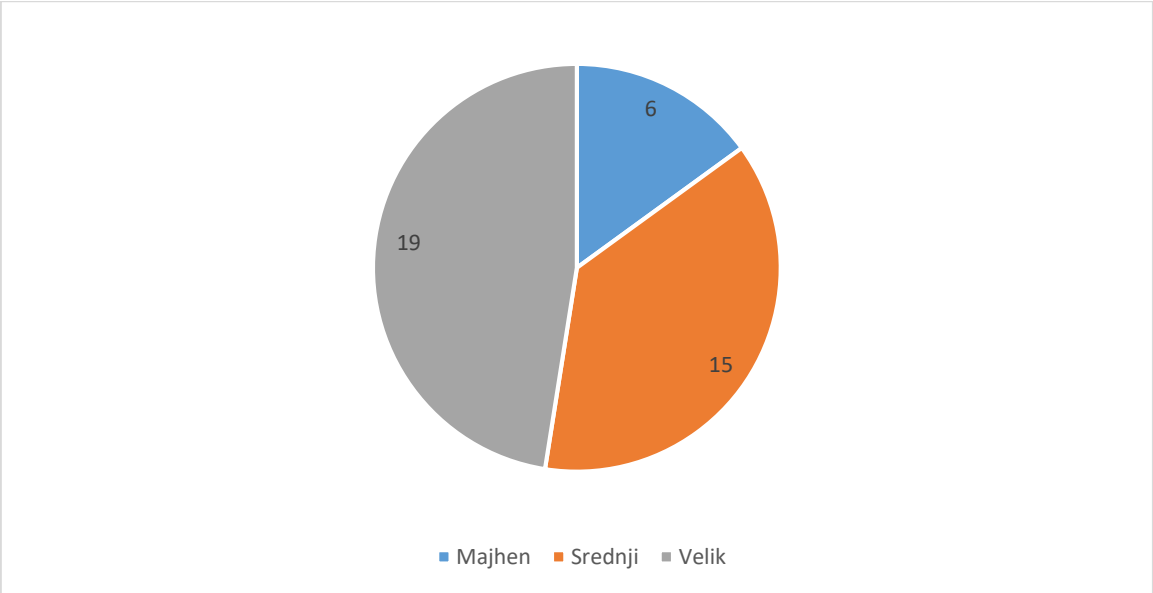
V tem poglavju smo opisali vsakega izmed izbranih odprtokodnih projektov ter zapisali področje, na katerem je aplikacija uporabljena. Pri izbiri projektov smo se odločili za izvorne kode programov, ki smo jih v prejšnjem poglavju opisali, če je bila le-ta na voljo. Nato smo med članki, ki smo jih analizirali pri sistematičnem pregledu literature, pogledali, katere projekte so analizirali avtorji člankov. Preostale projekte smo izbrali glede na sledeče kriterije, kjer smo izbrali skupno štirideset projektov:

- izvorna koda mora biti objavljena na eni izmed večjih repozitorijev za nalaganje projektov, kot je recimo Git in SourceForge,
- izvorna koda projekta mora biti napisana v programskem jeziku Java,
- orodje mora imeti dostopno izvorno kodo in se more uspešno prevesti v strojno kodo za potrebe analize z orodjem ckjm.

4.3 Podatki o projektih

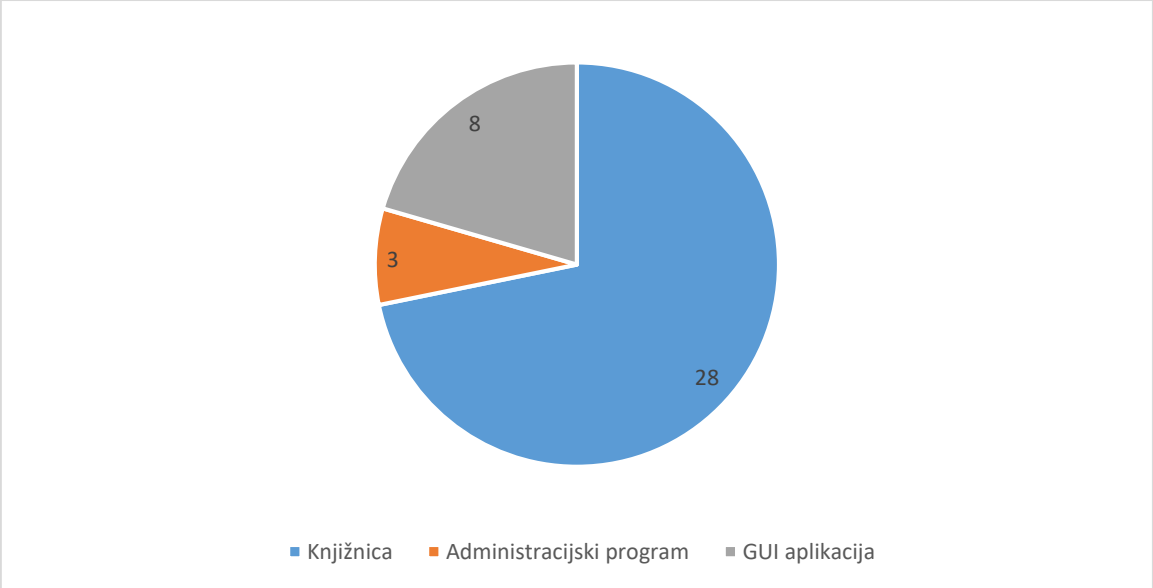
V Prilogi E so navedeni izbrani projekti ter podane poglavitne informacije o teh projektih, ki smo jih uporabili pri eksperimentalnem delu magistrskega dela. Pri tem so projekti v tabeli razvrščeni po abecednem vrstnem redu. V tabeli podane tudi informacije o verziji programske kode, ki smo jo uporabili pri analizi kode.

Ker smo za potrebe analize potrebovali večje število projektov, ki so se hkrati uspešno prevedli, smo projekte izbrali iz različnih virov. Tako smo kot prvo uporabili izvorno kodo programov za analizo kode, kjer je bila ta na voljo. V sledečem koraku izbora smo pregledali analizirane članke v sistematičnem pregledu literature ter izbrali projekte, ki so jih avtorji v teh člankih uporabili za analizo. Nato smo pregledali lestvice najbolj pogosto prenesenih projektov glede na trenutni dan. Kot zadnje smo pregledali najbolj pogoste in uspešne odprtokodne projekte ter izbrali tiste, ki so se uspešno prevedli ter smo jih tako lahko uporabili v naši analizi.



Slika 10: Velikost odprtokodnih projektov.

Kot lahko vidimo iz slike (glej Slika 10), smo izbrali projekte različnih velikosti, od manjših projektov do večjih projektov z več sto tisoč vrstic kode. Razlog izbire je, da mora metrika imeti pravilne rezultate tako za manjše kakor tudi za večje projekte. Kot je razvidno iz grafa, smo izbrali večje število večjih projektov, saj je rezultat določene metrike pri večjih projektih bolj reprezentativen, saj ima na voljo več podatkov in tako lahko bolj natančno napove vzdrževalnost.



Slika 11:Tip odprtokodnih projektov.

Kot je razvidno iz slike (glej Slika 11), smo tudi pri tipu projektov poizkusili izbrati različne tipe projektov, vendar smo zaradi projektov, ki večinoma obstajajo v odprtokodnem področju, zbrali predvsem knjižnice in aplikacije, saj v odprtokodnih projektih redkeje naletimo na administracijske programe, saj se le-ti pogosteje pojavljajo v poslovnem področju.

4.4 Izvedba eksperimenta

V tem poglavju smo opisali, kako je potekala sama izvedba eksperimenta ter kako smo prišli do rezultatov.

Kot smo že predstavili, smo se pri sistematičnem pregledu literature odločili za 5 različnih indeksov vzdrževalnosti, ki smo jih med seboj primerjali in poizkušali priti do ugotovitev. Te indekse smo natančneje opisali v poglavju 2.7. V naslednjem poglavju smo nato za izbrane indekse pregledali, katere vrednosti potrebujemo in katera orodja omogočajo pridobitev vrednosti teh metrik. Kot smo kasneje ugotovili, smo za analizo z orodjem ckjm potrebovali prevedeno strojno kodo orodja, medtem ko smo pri ostalih metričnih orodjih potrebovali le izvorno kodo.

V prejšnjem poglavju smo se nato dotaknili teme, kako smo izbrali projekte, ki smo jih analizirali in na podlagi katerih smo preverjali veljavnost metrik. Naleteli smo predvsem na problem, da večjega števila odprtokodnih projektov ni možno uspešno prevesti brez določenih popravkov oziroma sprememb, saj so le-ti veliki odprtokodni projekti in so zato večinoma slabo vzdrževani kot celota, ker je razvijalcem načeloma vseeno glede splošnega prevajanja projektov, saj pogosto potrebujejo le določen del projekta.

Ko smo uspešno našli in zagotovili zadostno število uspešno prevedenih odprtokodnih projektov, smo morali pridobiti rezultate metrik. Ker je način, kako deluje vsako izmed orodij različen, smo morali nato za vsako izmed orodij pognati ukaz v komandni vrstici. Ukaze, kako pridobiti rezultate metrik, smo natančneje opisali v tretjem poglavju. Za potrebe tega smo napisali preprost program, ki se je sprehodil skozi seznam programov ter izpisal ukaz za vsakega izmed izbranih analiznih programov. Ukaze smo nato uporabili in pridobili različne datoteke, v katerih so bili osnovni podatki o metrikah projektov.

V naslednjem koraku smo morali združiti metrike v skupno datoteko, kjer smo si pomagali s knjižnico Apache POI in združili podatke iz različnih projektov ter v različnih formatih v obliko, ki smo jo lahko nato uporabili. Pri tem je največji problem predstavljalo predvsem dejstvo, da so določeni podatki podani kot povprečje, drugi podatki pa kot posamezni podatek za vsak razred. Zato smo to spremenili v želeno obliko, ki smo jo nato zapisali v končno datoteko. Dobili smo rezultate, ki jih lahko vidimo v Tabela 16 in Tabela 17.

V naslednjem koraku smo nato iz pridobljenih vrednosti metrik zapisali formule ter pridobili vrednosti indeksov vzdrževalnosti. Končne rezultate za vrednosti različnih indeksov, ki smo jih nato uporabili za primerjavo, podaja tabela 18.

Kot zadnje smo v programu Microsoft Excel uporabili funkciji, ki izračunata vrednost obeh izmed izbranih statističnih metod za izračun korelacije med dvema spremenljivkama. Prišli smo do končnih rezultatov, ki smo jih analizirali ter natančneje opisali v naslednjem poglavju.

Tabela 16: Vrednosti metrik za izračun MI.

	LOC	*CC*	*HalE*	*HalV*
Activiti	72,80294	5,661412	193733	2820,915
Alfresco	139,7995	13,434	789240,8	5708,148
Ant	88,48227	10,57318	226580,7	3170,24
Apache-poi	106,0593	8,474911	303707,2	4343,769
Arduino	81,3311	9,29097	219421,3	3292,508
Ckjm	61,125	8,125	79809,53	2223,402
Commons-lang	149,9859	13,88283	1653284	8497,635
Couchbase	68,89696	4,105386	197552,7	2648,352
Ditaa	127,7067	20,82667	630803	6476,874
Drill	65,62929	6,158916	653141,2	2642,173
Easymock	54,48988	4,040486	98768,97	1779,717
Freemind	82,34352	7,811736	195196,3	3048,725
Gephi	90,18331	8,617785	286204,8	3686,374
Gifencoder	34,71875	2,875	82446,54	1429,989
Giraph	65,77521	4,490496	108956,9	2042,589
Guava	78,18468	5,911243	428157,9	3654,093
Hadoop	111,6188	7,971793	558390,9	4646,122
Hibernate	66,91261	3,784668	148960,9	2107,81
Inbloom	69,60976	5,658537	104967,7	2727,277
Ip-scan	44,6014	2,361305	62232,04	1468,925
Javageom	100,216	6,993827	391232,5	4763,876
Javancss	169,8056	43,81667	1644316	8333,18
Jdom	140,3454	16,64903	608770,7	5980,863
Jenkins	57,90545	5,336031	154563,3	2282,282
Jfreechart	149,1739	12,04666	522745,1	6070,605
Joda-time	191,6228	11,65179	1342523	11113,94
Jsoniter	54,20588	6,808824	181056,7	2216,384
Junit	26,04869	1,388577	31262,75	732,4777
LibGDX	96,0126	10,10119	453512,5	4621,501
Log4j	76,87056	7,073604	189359,9	2667,955
Neo4j	88,82414	4,227011	188150,2	2686,366
Pentaho	94,99252	8,083113	283790,6	3765,292
Slf4j	52,89407	4,050847	117877,3	1773,12
Sonar-qube	91,93958	5,847986	1715848	3759,002
Sonar-scanner	36,40816	1,673469	89162,74	1634,918
Spring	58,10871	4,232128	162880	2182,604
Spring-boot	40,39327	2,032807	58308,37	1240,734
Treeview3	101,4084	11,4802	336342,2	3870,898
Worldwind	144,6635	17,26555	849347,4	6175,482
Wro4j	72,56272	2,713262	88479,35	2169,613

Tabela 17: Rezultati vrednosti metrik za izračun MIwCK.

	D	*Ca*	*CC*	*Ce*	*CBO*	*RFC*
Activiti	0,13834	6,819728	9,177141	10,88234	6,613175	21,95196
Alfresco	0,217649	7,645754	16,27269	11,72786	7,499644	20,86892
Ant	0,172361	4,614682	17,21159	8,833604	4,353828	24,79687
Apache-poi	0,195081	5,654408	24,73051	10,14228	6,818778	26,85015
Arduino	0,222329	4,51028	16,01703	8,978147	4,176056	23,26291
Ckjm	0,112	2,9	32	7	6,444444	29,11111
Commons-lang	0,183116	2,809436	42,91455	10,1296	2,384058	23,24493
Couchbase	0,219247	5,950206	14,86034	15,33956	6,151625	14,26354
Ditaa	0,207794	4,546857	16,38277	8,970689	3,52	34,74667
Drill	0,22324	13,4874	35,7365	24,9575	10,12543	30,11092
Easymock	0,186746	4,14154	18,71821	8,943967	3,027864	15,89474
Freemind	0,204275	4,7293	18,30833	9,454455	5,39	23,61909
Gephi	0,192588	5,019332	26,5235	10,96104	5,240122	19,55623
Gifencoder	0,297649	7,645754	49,27269	18,72786	2,638889	11,91667
Giraph	0,20883	4,655163	43,1229	9,877764	5,694585	16,65415
Guava	0,19089	4,416455	185,9811	14,10474	3,964046	14,38004
Hadoop	0,118567	1,893076	9,47943	14,79146	4,234043	21,56028
Hibernate	0,308404	3,372307	5,911997	8,327005	10,5625	25,25
Inbloom	0,319873	2,689176	5,927822	11,56337	4,648649	17,51351
Ip-scan	0,246755	4,757583	19,09747	8,434214	5,311067	22,46162
Javageom	0,244275	4,7293	40,30833	12,45446	9,485597	32
Javancss	0,196584	1,983026	15,98178	7,337615	4,227273	33
Jdom	0,199755	4,880885	14,43962	8,467039	5,863415	22,99512
Jenkins	0,178933	5,94563	24,2593	17,89965	5,520108	18,78809
Jfreechart	0,185172	4,726263	10,13131	9,59596	6,529954	36,03379
Joda-time	0,237578	4,981043	59,96695	13,28429	8,801512	46,15501
Jsoniter	0,43312	4,5774	63,20224	13,77847	2,669355	9,137097
Junit	0,190806	5,409457	19,97249	8,705132	2,480769	9,088018
LibGDX	0,261097	7,345431	19,85662	7,636284	4,996989	26,817
Log4j	0,13492	3,660726	17,47108	9,932859	4,179412	23,1
Neo4j	0,190452	1,405925	94,36232	7,816521	9,445958	15,13533
Pentaho	0,229665	5,323165	12,45701	9,517371	6,313233	24,74707
Slf4j	0,169932	3,503064	8,712025	8,080491	2,747036	17,36364
Sonar-qube	0,216241	7,154799	14,40449	12,55694	8,492008	24,84909
Sonar-scanner	0,248861	0,994342	12,99393	9,112538	4,173913	21,69565
Spring	0,229108	6,824229	10,49475	12,94356	5,281552	18,54491
Spring-boot	0,213259	5,969551	11,77972	10,75764	4,234024	12,86557
Treeview3	0,193214	3,312228	15,79353	9,861243	3,47205	23,6501
Worldwind	0,189945	5,930344	17,97101	12,96346	5,882961	26,39847
Wro4j	0,200626	7,848346	12,73557	16,27434	5,603565	15,79032

Tabela 18: Vrednosti indeksov vzdrževalnosti.

	MI	*MIv2*	*MIwLOC*	*MIvMartins*	*MIvCK*
Activiti	59,70137	59,78073	56,91209	45,61799	43,78985
Alfresco	42,69882	43,88865	39,32844	42,13874	42,26964
Ant	54,91842	54,92321	51,84709	47,59409	47,70136
Apache-poi	51,50581	50,869	47,01355	44,72592	45,68754
Arduino	56,67087	56,36972	54,0509	47,505	47,24097
Ckjm	64,94736	63,24919	61,36098	48,76776	47,11767
Commons-lang	38,95616	40,59107	37,29627	49,40832	49,34946
Couchbase	60,87518	61,34917	58,32329	50,23607	40,9751
Ditaa	43,20794	42,97893	41,89445	47,54593	53,03417
Drill	57,11463	61,66644	59,56037	36,94949	43,20734
Easymock	67,00062	67,18456	64,25074	48,49254	45,37709
Freemind	57,21092	56,91201	53,72894	47,3005	45,96295
Gephi	54,26923	54,28393	51,34552	46,92125	44,36449
Gifencoder	75,09459	75,80208	75,47394	42,8085	44,09806
Giraph	63,55169	63,35304	59,50395	44,02179	42,55956
Guava	55,80659	57,2365	55,07434	27,01615	43,60954
Hadoop	48,73349	49,81736	45,62367	62,36101	46,42844
Hibernate	62,37643	63,07762	59,06801	50,17958	40,56694
Inbloom	62,50327	60,67455	58,06049	54,94587	44,17035
Ip-scan	72,16133	71,77277	69,24607	45,52724	45,55036
Javageom	51,89217	51,63632	48,54059	46,29938	44,78851
Javancss	30,10406	31,8221	33,62127	52,77085	51,43559
Jdom	42,77975	42,84415	39,21673	46,60178	45,13118
Jenkins	64,20731	64,62049	62,72456	51,85559	43,69815
Jfreechart	43,38018	42,84915	37,45435	48,96727	50,04189
Joda-time	36,25742	35,78866	29,91976	43,52639	51,78215
Jsoniter	64,38702	65,49045	64,38165	41,87702	42,8474
Junit	83,33053	84,2197	82,70151	44,74285	43,04868
LibGDX	51,36081	51,76499	49,68561	37,7649	47,82458
Log4j	58,58433	58,87595	55,51251	51,59966	47,16581
Neo4j	56,94827	57,18234	51,74563	42,73969	37,46546
Pentaho	53,58974	53,46549	49,96981	46,38545	45,36554
Slf4j	66,87249	67,67707	64,99491	51,06435	46,35066
Sonar-qube	48,50844	54,51093	50,83607	44,52174	42,83699
Sonar-scanner	74,34447	74,62181	74,29093	56,79965	46,55861
Spring	64,22694	65,05054	62,63642	46,17784	43,87362
Spring-boot	74,04394	74,31186	71,70891	46,2614	42,62887
Treeview3	51,18505	51,4946	48,2231	51,8887	48,2416
Worldwind	41,0208	42,05098	38,34453	47,88517	46,59535
Wro4j	63,09695	61,87675	56,9969	46,84068	42,28956

4.5 Statistična analiza zbranih rezultatov

V tem poglavju smo predstavili rezultate, ki smo jih dobili pri uporabi Pearsonove korelacije in F-Testu med metrikami. Rezultate smo analizirali za vsako metriko posebej. V tabeli (glej Tabela 19) lahko vidimo pridobljene rezultate.

Tabela 19: Rezultate koeficient korelacije.

Indeks vzdrževalnosti	*Koeficient korelacije*
MIv2	0,992919
MIwLOC	0,983572
MIvMartins	-0,01034
MIvCK	-0,52056

Kot smo že zapisali, smo izračunali koeficient korelacije v primerjavi med indeksom vzdrževalnosti in ostalimi indeksi. Iščemo vrednost, ki se čim bolj približa vrednosti 1 oziroma -1, saj bi to pomenilo, da lahko glede na gibanje vrednosti ene metrike dokaj natančno napovemo, kako se bo gibala druga metrika, kar pomeni, da lahko glede na že validirane mejne vrednosti za vzdrževalnost programske opreme ugotovimo, ali določena metrika napove, da bo programska oprema lažja oziroma težja za vzdrževanje.

Pričakovano izboljšan indeks vzdrževalnosti MIv2 doseže skoraj popolno vrednost 1, saj so med indeksoma le manjše razlike. MIv2 je predstavljen kot izboljšava osnovnega indeksa in je zato takšen rezultat pričakovan.

Indeks vzdrževalnosti z uporabo števila vrstic kode MIwLOC doseže zelo dober rezultat, saj se približa vrednosti 1 na le 2 odstotni točki, kar pomeni, da lahko dokaj natančno napovemo vzdrževalnost programske opreme z uporabo MIwLOC. Če je vrednost MI pri določenem projektu višja, bo ta verjetno višja tudi pri metriki MIwLOC.

Vrednost metrike MIvMartins, ki je dosegla rezultat -0,01, kar je skoraj najslabši možen rezultat, pove, da je gibanje metrike skoraj povsem naključno in da se vrednost MI glede na vrednosti MIvMartins spreminja naključno. Iz tega lahko zaključimo, da metrike ne moremo uporabiti za napovedovanje vzdrževalnosti.

Zadnja uporabljena metrika MIvCK je dosegla rezultat -0,52, kar pomeni da je določena stopnja ujemanja med metrikama, vendar bi le z uporabo MIvCK težko napovedali točno stanje vzdrževalnosti določene programske opreme. Kljub temu je metrika v primerjavi z MIvMartins dosegla rezultat, ki nakazuje, da lahko vsaj delno uporabimo metriko za napovedovanje vzdrževalnosti, vendar se točnost napovedi ne more primerjati z metrikama MIv2 in MIwLOC.

Na prejšnji strani smo predstavili rezultate pri analizi z uporabo korelacijskega koeficienta. V spodnji tabeli (glej Tabela 20) lahko vidimo rezultate pri uporabi statistične metode F-Test za izračun variance. Natančnejši rezultati so v prilogi I.

Tabela 20: F-Test rezultati.

Indeks vzdrževalnosti	*Rezultati F-test*
MIv2	1,026539
MIwLOC	0,943901
MIvMartins	4,129906
MIvCK	13,52324

F-Test za izračun variance med dvema spremenljivkama se uporablja za ugotavljanje podobnosti porazdelitve variance med dvema spremenljivka. Kot smo že zapisali, je želena vrednost 1, kar bi pomenilo, da je podobnost razdelitve variance med dvema spremenljivkama enaka. Tako kot pri izračunu z uporabo korelacije smo tudi tu kot referenco uporabili osnoven indeks vzdrževalnosti.

Metrika MIv2 je dosegla rezultat 1,026, kar pomeni, da lahko s pomočjo razlike med dvema izbranima točkama pri metriki MI na podlagi gibanja vzdrževalnosti ugotovimo, kakšna je bila sprememba v vzdrževalnosti pri metriki, saj je porazdelitev vzorca med obema metrika podobna.

Presenetljivo je tudi v tem kriteriju metrika MIwLOC dosegla dober rezultat, saj se je z 0,94 dovolj približala porazdelitvi MI, da lahko potrdimo ujemanje variance. To pove, da se vrednost metrike giblje podobno, kot se giblje vrednost indeksa vzdrževalnosti. To tudi pove, da so rezultati projektov, ki smo jih uporabili, porazdeljeni podobno v različne skupine, kot bi bili porazdeljeni, če bi uporabili indeks vzdrževalnosti.

Kot že pri prejšnjih rezultatih, sta se preostali dve metriki tukaj odrezali slabše od ostalih indeksov. Metrika MIvMartins je dosegla rezultat 4,13, kar pomeni, da bi lahko z njeno uporabo približno napovedali varianco gibanja določene metrike, vendar bi rezultat vsekakor lahko bil bližje vrednosti 1.

Metrika MIvCK je dosegla rezultat 13,5, kar pomeni, da je varianca gibanja spremenljivke v primerjavi z indeksom vzdrževalnosti različna. Ker metrika nima praktične uporabe pri napovedovanju vzdrževalnosti, ne moremo zapisati meje, glede na katero bi napovedali vzdrževalnost programske opreme.

Prav tako ostali kriteriji pri F-Testu, kot je recimo interval zaupanja, podajo podobne rezultate, kot smo jih zapisali ter tako predstavijo metriki MIv2 in MIwLOC kot boljši možnosti.

4.6 Interpretacija rezultatov

V tem poglavju smo povezali rezultate, ki smo jih dobili pri eksperimentalnem delu z uporabo obeh testov ter zapisali končne ugotovitve.

Kot smo že ugotovili, sta metriki MIvMartins in MIvCK dosegli slabše rezultate v obeh statističnih testih, predvsem pri uporabi F-Testa. Rezultati so takšni, da lahko trdimo, da z uporabo katere izmed teh metrik ni mogoče natančno napovedovati vzdrževalnosti. Možnosti, zakaj smo prišli do drugačnih ugotovitev kot avtor osnovnega besedila, je več. Najverjetnejša je ta, da smo uporabili večji vzorec podatkov kot avtor članka in tako pridobili natančnejše rezultate, saj je avtor pri analizi vzdrževalnosti uporabil le pet različnih odprtokodnih projektov, ki so vsi približno enake velikosti, in podoben tip aplikacij.

Vrednost statističnih testov pri izboljšanem indeksu vzdrževalnosti MIv2 je pričakovana, saj doseže skoraj optimalne vrednosti in jo lahko tako uporabimo za napovedovanje vzdrževalnosti. Prav tako je samo gibanje vrednosti precej podobno gibanju vrednosti osnovnega indeksa in so tako najverjetneje mejne vrednosti obeh indeksov enake.

Presenetljivo dober rezultat je dosegla tudi poenostavljena oblika osnovnega indeksa MIwLOC, ki se je zelo približal rezultatom, ki jih je dal MI in bi ga lahko uporabili za napovedovanje vzdrževalnosti, saj je tako korelacija z indeksom MI kakor tudi varianca med indeksoma podobna.

Iz primerjave enačb MIv2 in MIwLOC lahko ugotovimo, da metrika MIv2 potrebuje več različnih vrednosti za njen izračun, saj potrebuje tako število vrstic kode, ciklomatično kompleksnost kot tudi volumen, medtem ko MIwLOC za svoj izračun potrebuje le število vrstic kode in nato lahko na podlagi formul, ki so bile natančneje zapisane v [26], pridobimo dovolj natančen približek za vrednost prej napisanih spremenljivk. Končni rezultat je tako dovolj dober, da lahko z njegovo uporabo napovemo težavnost vzdrževanja izvorne kode. Posledica tega je, da je enačba za MIwLOC lažje izračunljiva ter tudi bolj pregledna, saj je vrednost LOC pri uporabi različnih orodij med seboj skoraj enaka.

Lahko zaključimo, da je indeks MIwLOC dovolj dober, da lahko z njim napovemo vzdrževalnost programske opreme, saj za izračun indeksa potrebuje le povprečno število vrstic kode programske opreme in tudi predlagamo njegovo uporabo, saj je zaradi preprostejše formule izračuna indeksa manjša razlika med rezultati, ki jih dobimo, če uporabimo različna orodja za analizo izvorne kode projektov.

4.7 Omejitve eksperimenta in tveganja za pravilnost

Pri izdelavi magistrske naloge smo naleteli na več dejavnikov, ki bi lahko vplivali na pravilnost rezultatov magistrske naloge ter tudi določene pomanjkljivosti uporabljenih tehnik. V tem poglavju smo zapisali te težave ter jih pojasnili.

Največje tveganje za pravilnost rezultatov eksperimentalnega dela magistrske naloge je različna vrednost metrik med orodji na istih projektih, kot so že ugotovili določeni članki – [24] in [26]. Problem je predvsem v tem, da različna orodja uporabljajo definicije za določene metrike, ki niso dovolj natančno določene in tako je njihov način izračuna vrednosti metrik različen. Pri preizkusu različnih orodij smo tudi sami to ugotovili, vendar smo hkrati tudi ugotovili, da se pri večjih programskih opremah vrednosti dokaj približajo. Temu tveganju smo se poizkusili izogniti tako, da smo za določen indeks vzdrževalnosti uporabili le eno orodje, s katerim smo pridobili vse potrebne vrednosti za izračun določene metrike. Kljub temu še vedno obstaja možnost, da so bile pridobljene vrednosti drugačne od teh, ki bi jih pridobili z drugim orodjem.

Naslednji problem je ta, da so enačbe za metrike v samih člankih slabo definirane ter ni jasno določeno, katere vrednosti metrik so bile uporabljene ter na kateri ravni so vzeli povprečje, saj je možno uporabiti povprečje na ravni paketa in nato izračunati povprečje celotne izvorne kode, možno pa je tudi uporabiti povprečje celotne izvorne kode. Pri izvajanju praktičnega magistrske naloge se temu tveganju nismo mogli izogniti, vendar smo pri uporabi različnih formul za izračun povprečja in drugih vrednosti metrik ugotovili, da prihaja pri končnem izračunu do le manjših razlik med vrednostmi, ki naj ne bi imele bistvenega vpliva na končen rezultat.

Po izbiri metrik, ki smo jih uporabili v praktičnem delu magistrskega dela, smo zasledili tudi določene članke, kjer se avtor sprašuje, ali je indeks vzdrževalnosti res primeren za napovedovanje vzdrževalnosti [36]. Problem je predvsem v tem, da je formula indeksa bila izračunana na podlagi matematične enačbe. Tako vrednosti parametrov nimajo praktičnega pomena, ker so bile ugotovljene na podlagi analize obstoječih projektov. Težava je, da je nemogoče predvideti, kako bo sprememba programske kode vplivala na vrednost indeksa vzdrževalnosti. Problem je tudi, da smo pri uporabi vseh indeksov vzdrževalnosti vzeli povprečje celotne programske opreme, kar lahko pomeni, da se lahko v izvorni kodi programske opreme skrivajo določeni deli, ki imajo zelo slabe rezultate, vendar jih zaradi uporabe povprečja ne zaznamo. To lahko vodi do tega, da imamo določene dele programske opreme, ki jih je praktično nemogoče vzdrževati, čeprav je indeks vzdrževalnosti na nivoju sistema takšen, da bi morala biti koda lahko vzdrževana. Dodaten problem je tudi starost indeksa vzdrževalnosti, saj se je zaradi zapisanih težav področje raziskav preusmerilo na druge tehnike napovedovanja vzdrževalnosti [37].

5 SKLEP

V sklopu magistrskega dela smo izvedli sistematičen pregled literature, izbrali določene metrike za napovedovanje vzdrževalnosti, ki temeljijo na indeksu vzdrževalnosti, zbrali orodja ter z njihovo uporabo izračunali vrednost metrik. Rezultate smo nato med seboj primerjali ter poizkušali ugotoviti, ali obstaja nadomestilo za indeks vzdrževalnosti, ki je, glede na ideje nekaterih avtorjev, nekoliko zastarel, saj pri svoji analizi ne uporablja objektno-orientiranih metrik za izračun vzdrževalnosti kode.

Dobljeni rezultati kažejo na to, da v literaturi še ni bila podana primerna ideja oziroma formula, kjer bi lahko indeks vzdrževalnosti zamenjal nov, ki bi v svoji enačbi uporabil objektno-orientirane metrike. Kljub temu smo v sklopu magistrske naloge ugotovili, da obstaja preprostejša metrika, ki poda podobne rezultate kot indeks vzdrževalnosti, za njen izračun pa ne potrebujemo toliko vrednosti metrik.

Ideja, da vrednost parametrov nadomestimo z enačbo, ki je splošno dokazano veljavna za manjše programe, je dovolj dober približek, da lahko napovemo tudi vzdrževalnost večje programske opreme. Tako lahko z uporabe metrike iz [26], kjer potrebujemo le povprečno število vrstic modulov v kodi, dovolj natančno napovemo, kako težko bo vzdrževanje programske opreme. Prednost tega je predvsem v tem, da se s tem izognemo problemom pri pridobivanju vrednosti ostalih metrik, saj je metrika LOC najbolj pogosta in je pridobivanje njene vrednosti dokaj preprosto.

Kot smo zapisali v tveganjih za raziskavo, indeks vzdrževalnosti v praksi ni tako uporaben, kot se mogoče zdi na prvi pogled. Problem je predvsem v tem, da ne vemo, kako točno je metrika izračunana, saj kot je razvidno iz formul, so bili uporabljeni parametri za pomembnost določene spremenljivke izračunani s pomočjo linearne regresije ter so tako nastali kot posledica matematične analize. V praksi to pomeni, da razvijalec pri razvoju programske opreme ne more vedeti, kako bo dodajanje kode vplivalo na vrednost indeksa. Indeks vzdrževalnosti je lahko bolj uporabljen kot nekakšno merilo, ki pove, kako se giblje vzdrževalnost programske opreme skozi čas, ne moremo pa ga uporabiti za napovedovanje pri manjših spremembah kode. Lahko se zgodi, da je nova koda lažje vzdrževana, vendar je kljub temu vrednost indeksa vzdrževalnosti nižja kot pred spremembami. Predlagamo, da se indeks vzdrževalnosti uporabi predvsem kot kontrolna vrednost in se določi nekakšno želeno vrednost za našo programsko opremo. V kolikor se vrednost preveč odmakne od želene vrednosti, se sprožijo določeni procesi, kjer v naslednjih iteracijah programske kode več časa posvetimo izboljšanju vrednosti indeksa in tako vzdrževalnost dolgoročno ohranimo na sprejemljivem nivoju.

LITERATURA IN VIRI

[1] L. Erlikh, “Leveraging Legacy System Dollars for E-Business,” *IT Pro*, no. June, pp. 17–23, 2000.

[2] D. Coleman, D. Ash, B. Lowther, and P. Oman, “Using Metrics to Evaluate Software System Maintainability,” *Comput. Pract.*, vol. 27, no. 8, pp. 44–49, 1994.

[3] G. Vale, R. Abilio, A. Freire, and H. Costa, “Criteria and Guidelines to Improve Software Maintainability in Software Product Lines,” *2015 12th Int. Conf. Inf. Technol. - New Gener.*, pp. 427–432, 2015.

[4] M. K.G., “Empirical comparison of two metric suites for maintainability prediction in packages of object-oriented systems: A case study of open source software,” *J. Comput. Sci.*, vol. 10, no. 11, pp. 2330–2338, 2014.

[5] A. April and A. Abran, *Software Maintenance Management: Evaluation and Continuous Improvement*. 2008.

[6] D. Le, J. Xiao, H. Huang, H. Wang, C. William, and N. York, “Shadow Patching : Minimizing Maintenance Windows in a Virtualized Enterprise Environment,” in *The 10th International Conference on Network and Service Management (CNSM)*, 2014, pp. 169–174.

[7] G. Scanniello, “Source Code Survival with the Kaplan Meier Estimator,” in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, pp. 524–527.

[8] W. L. Miller, D. Ph, and L. B. Compton, “Assuming Software Maintenance of a Large , Embedded Legacy System from the Original Developer,” in *2013 IEEE International Conference on Software Maintenance Assuming*, 2013, pp. 552–555.

[9] W. Zhang, L. Huang, and V. Ng, “SMPLearner : learning to predict software maintainability,” *Autom. Softw. Eng.*, vol. 22, no. 1, pp. 111–141, 2015.

[10] B. Kitchenham, “Procedures for Performing Systematic Reviews,” 2004.

[11] B. Kitchenham, “Guidelines for performing Systematic Literature Reviews in Software Engineering.”

[12] K. D. Welker, P. W. Oman, and G. G. Atkinson, “Development and Application of an Automated Source Code Maintainability Index,” *Softw. Maint. Res. Pract.*, vol. 9, no. March, pp. 127–159, 1997.

[13] R. K. Bandi, V. K. Vaishnavi, and D. E. Turk, “Object-Oriented Design Complexity Metrics,” *IEEE Trans. Softw. Eng.*, vol. 29, no. 1, pp. 77–87, 2003.

[14] M. G. Bocco, D. L. Moody, and M. Piattini, “Assessing the capability of internal metrics as early indicators of maintenance effort through experimentation,” *J. Softw. Maint. Evol. Res. Pract.*, no. April 2004, pp. 225–246, 2005.

[15] C. Van Koten and A. R. Gray, “An application of Bayesian network for predicting object-oriented software maintainability,” *Inf. Softw. Technol.*, vol. 48, pp. 59–67, 2006.

[16] I. Heitlager, T. Kuipers, and J. Visser, “A Practical Model for Measuring Maintainability,” *6th Int. Conf. Qual. Inf. Commun. Technol. (QUATIC 2007)*, pp. 30–39, 2007.

[17] Y. Zhou and H. Leung, “Predicting object-oriented software maintainability using multivariate adaptive regression splines,” *J. Syst. Softw.*, vol. 80, pp. 1349–1361, 2007.

[18] Z. Yuming and X. Baowen, “Predicting the maintainability of open source software using design metric,” *Wuhan Univ. J. Nat. Sci.*, vol. 13, no. 1, 2008.

[19] T. Abd El-lateef, A. H. Yousef, and M. F. Ismail, “Object oriented design metrics framework based on code extraction,” in *Computer Engineering Systems, 2008. ICCES 2008. International Conference on*, 2008, pp. 291–295.
[20] N. Upadhyay, B. M. Deshapande, and V. P. Agrawal, “Developing Maintainability Index of a Software Component : A Digraph and Matrix Approach,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 35, no. 5, 2010.
[21] T. R. G. Nair, S. Aravindh, and R. Selvarani, “Design Property metrics to Maintainability estimation – A virtual method using functional relationship mapping,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 35, no. 6, pp. 1–6, 2010.
[22] K. Kaur and H. Singh, “Determination of Maintainability Index for Object Oriented Systems,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 36, no. 2, pp. 1–6, 2011.
[23] T. Bakota and P. Heged, “A Cost Model Based on Software Maintainability,” in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, 2012, pp. 316–325.
[24] N. Ramasubbu, C. F. Kemerer, and J. Hong, “Structural Complexity and Programmer Team Strategy : An Experimental Test,” *IEEE Trans. Softw. Eng.*, vol. 38, no. 5, pp. 1054–1068, 2012.
[25] J. Al Dallal, “Object-oriented class maintainability prediction using internal quality attributes,” *Inf. Softw. Technol.*, vol. 55, no. 11, pp. 2028–2048, 2013.
[26] N. M. A. M. Najm, “Measuring Maintainability Index of a Software Depending on Line of Code Only,” *IOSR J. Comput. Eng.*, vol. 16, no. 2, pp. 64–69, 2014.
[27] H. W. Alomari, M. L. Collard, and J. I. Maletic, “A Slice-Based Estimation Approach for Maintenance Effort,” in *2014 IEEE International Conference on Software Maintenance and Evolution*, 2014, no. 3, pp. 82–90.
[28] L. Kumar, D. Kumar, and S. K. Rath, “Validating the Effectiveness of Object-Oriented Metrics for Predicting Maintainability,” *Procedia - Procedia Comput. Sci.*, vol. 57, pp. 798–806, 2015.
[29] S. Mishra, “Maintainability Prediction of Object Oriented Software by using Adaptive Network based Fuzzy System Technique,” *Int. J. Comput. Appl.*, vol. 119, no. 9, pp. 24–27, 2015.
[30] S. Almugrin, W. Albattah, and A. Melton, “Using indirect coupling metrics to predict package maintainability and testability,” *J. Syst. Softw.*, vol. 121, pp. 298–310, 2016.
[31] N. Kayarvizhy, “Systematic Review of Object Oriented Metric Tools,” *Int. J. Comput. Appl.*, vol. 135, no. 2, pp. 8–13, 2016.
[32] D. Spinellis, “Tool writing: A forgotten Art?,” *IEEE Softw.*, vol. 22, no. 4, pp. 9–11, 2005.
[33] J. B. et Al., *Noise Reduction in Speech Processing*, vol. 2. 2009.
[34] G. W. Snedecor and W. G. Cochran, *Statistical methods*. 1967.
[35] M. I. Sarwar, W. Tanveer, I. Sarwar, and W. Mahmood, “A comparative study of MI tools: Defining the roadmap to MI tools standardization,” *IEEE INMIC 2008 12th IEEE Int. Multitopic Conf. - Conf. Proc.*, pp. 379–385, 2008.
[36] D. I. K. Sjøberg, B. Anda, and A. Mockus, “Questioning software maintenance metrics: A comparative case study,” *Int. Symp. Empir. Softw. Eng. Meas.*, no. Mi, pp. 107–110, 2012.
[37] R. Kazman, Y. Cai, R. Mo, Q. Feng, and L. Xiao, “A Case Study in Locating the Architectural Roots of Technical Debt,” in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2015, pp. 179–188.

Priloga A: Evaluacijska forma

	Besedilo
Splošne informacije	
ID študije	
Naslov	
Avtor	
Leto izdaje	
Založba	
Država izdaje	
Tip študije	
Opravljena revizija?	
Vzdrževalnost	
Definicija vzdrževalnosti	
Uporaba standarda ISO	
Metrika	
Ime metrike	
Formula	
Uporabljene metrike	
Tip uporabljenih metrik	
Nivo delovanja	
Projekti	
Programski jezik	
Tip programskega jezika	
Velikost projektov	
Število projektov	
Tip projektov	
Orodje	
Ime orodja	
Dostopnost orodja	
Rezultati	
Možne vrednosti metrike	
Način evaluacije rezultatv	

Priloga B: Metrike v sistematičnem pregledu literature

Metrika\študija	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21
Tradicionalne	x	x			x	x	x		x			x	x	x	x	x	x		x	x	x
LOC	x					x			x				x		x	x	x			x	
CC	x					x			x												
Halstead kompleksnost	x					x						x									
Delež komentarjev	x																				
Določene vrednosti za tip spremenljivke		x																			
SIZE2 (Number of properties in class)					x		x												x		
CHANGE (Number of lines changed in the class)					x		x						x		x		x		x		x
Sklopljenost razredov						x			x			x		x							
Vezljivost razredov						x						x		x	x						
Odstotek velikosti dela programa																	x				
Ponavljanje kode						x															
OO			x		x		x	x			x	x			x			x	x		x
MaxDIT (Maximum depth of inheritence tree)			x					x													
MaxHAgg (Maximum height of aggregation)			x																		
NOH (Number of generalization heracrchies)			x																		
NAggH (Number of aggregation hierarchies)			x																		
DIT (Depth of the inheritence tree)					x		x	x	x		x								x		
NOC (Number of children)					x		x	x			x								x		
MPC (Message-passing couple)					x		x												x		
RFC (Response for a class)					x		x	x			x							x	x		
LCOM (Lack of cohesion of methods)					x		x	x			x								x		
DAC (Data abstraction coupling)					x		x												x		
WMC (Weighted method per class)					x		x	x			x								x		
NOM (Number of methods)					x		x	x				x			x				x		
NPAVGC (Number of parameters per method)								x													
CSA (Average number of attributes per class)								x							x						
NCLASS (Number of classes)								x				x									
Coupling between objects											x	x						x			
Cohesion																					
A (Abstractness)												x						x			x
Nesting												x									
I (Instability of class)												x						x			x
Metrike načrtovanja			x	x																	
Število razredov v UML			x	x																	
Število metod v UML			x	x																	
Število atributov v UML			x	x																	
Število asociacij v UML			x	x																	
Število odvisnosti v UML			x	x																	
Število generalizacij v UML			x	x																	
Procesne													x								
C (Strošek sprememb v sistemu)													x								

Priloga C: Izbrani projekti v sistematičnem pregledu literature

ID ŠTUDIJE	PROGRAMSKI JEZIK	ŠTEVILO PROJEKTOV	VELIKOST PROJEKTOV	TIP PROJEKTOV	IZVORNA KODA
S1	Proceduralni (C)	Nizko	Veliki	Ni podatka	Ni dostopna
S2	Proceduralni (FORTRAN in C++)	Srednje	Različno	Različno	Ni dostopna
S3	Objektni (Ni podatka)	Srednje	Ni podatka	Ni podatka	Ni dostopna
S4	Modeling (UML)	Srednje	Srednji	Ni podatka	Ni dostopna
S5	Objektni (ADA)	Nizko	Majhni	Administratorski sistemi	Ni dostopna
S6	Ni podatka	Ni podatka	Ni podatka	Ni podatka	Ni dostopna
S7	Objektni (ADA)	Nizko	Majhni	Administratorski sistemi	Ni dostopna
S8	Objektni (Java)	Visoko	Različno	Različno	Dostopno
S9	Objektni (Ni podatka)	Nizko	Ni podatka	Administratorski sistemi	Ni dostopna
S10	Ni podatka	Nizko	Ni podatka	GUI aplikacije	Ni dostopna
S11	Objektni (Ni podatka)	Nizko	Srednji	Ni podatka	Ni dostopno
S12	Objektni (Java)	Srednje	Srednji	GUI aplikacije	Dostopno
S13	Objektni (Java)	Srednje	Veliki	Ni podatka	Delno dostopna
S14	Objektni (Java)	Nizko	Veliki	Administratorski sistemi	Ni dostopno
S15	Objektni (Java)	Srednje	Veliki	GUI aplikacije	Dostopno
S16	Objektni (C++)	Srednje	Majhni	Knjižnice	Ni dostopna
S17	Proceduralni (C in Assembly)	Nizko	Veliki	GUI aplikacije	Delno dostopna
S18	Objektni (Java)	Srednje	Majhni	GUI aplikacije	Dostopno
S19	Objektni (ADA)	Nizko	Majhni	Administratorski sistemi	Ni dostopna
S20	Objektni (ADA)	Nizko	Majhni	Administratorski sistemi	Ni dostopna
S21	Objektni (Java)	Srednje	Različno	Knjižnice	Dostopna

Priloga D: Ugotovitve sistematičnega pregleda literature

ID študije	Način rezultatov	Metrike, ki jih primerja
S1	Ne primerja	
S2	Primerjava z realnimi spremembami	MI
S3	Primerjava z realnim časom	Čas
S4	Primerjava z realnim časom	Čas
S5	Primerjava z realnimi spremembami	Število sprememb
S6	Ni primerjave	
S7	Primerja s realnimi spremembami	Število sprememb
S8	Primerja z drugimi metrikami	MI
S9	Primerja z drugimi metrikami	MI
S10	Ni primerjave	
S11	Primerja z realnimi spremembami	Število sprememb
S12	Primerja z drugimi metrikami	MI
S13	Primerja z realnimi spremembami	Strošek
S14	Primerja z realnimi spremembami	Čas, ocena težavnosti
S15	Primerjava z drugimi metrikami	Različne OO metrike
S16	Primerjava z drugimi metrikami	MI
S17	Primerjava z drugimi metrikami	Število sprememb
S18	Primerjava z drugimi metrikami	MI
S19	Primerjava z drugimi metrikami	Metrike z področja UI
S20	Primerja z realnimi spremembami	Različne OO metrike
S21	Primerja z realnimi spremembami	Število sprememb

Priloga E: Projekti pri napovedovanju vzdrževalnosti

Ime projekta	*LOC*	*Tip projekta*	*Izbira*	*Velikost*	*Verzija*
Activiti	203193	Knjižnica	Najdeno	Visoko	5.23.0
alfresco	967972	Administracija	Najdeno	Visoko	5.2
ant	137236	Knjižnica	Članki	Visoko	1.9.7
apache-poi	359329	Knjižnica	Najdeno	Visoko	3.16
arduino	24318	Knjižnica	Najdeno	Srednje	1.6.13
ckjm	489	Knjižnica	Orodje	Nizko	1.9
commons-lang	74243	Knjižnica	Najdeno	Srednje	3.5
couchbase	29419	Knjižnica	Najdeno	Srednje	1.3.6
ditaa	9578	GUI aplikacija	Najdeno	Nizko	0.1
drill	690092	Knjižnica	Najdeno	Visoko	1.4
easymock	13459	Knjižnica	Najdeno	Srednje	3.4
freemind	67357	GUI aplikacija	Članki	Srednje	1.0.1
gephi	115615	GUI aplikacija	Najdeno	Visoko	0.9.1
gifencoder	1111	Knjižnica	Najdeno	Nizko	0.9.0
giraph	159176	Knjižnica	Najdeno	Visoko	1.2.0
guava	251051	Knjižnica	Najdeno	Visoko	20.0
hadoop	1214859	Knjižnica	Najdeno	Visoko	2.6.5
hibernate	630183	Knjižnica	Najdeno	Visoko	5.2.5
inbloom	2854	Knjižnica	Najdeno	Nizko	2.0
ip-scan	19134	GUI aplikacija	Najdeno	Srednje	3.50.0
javageom	32470	Knjižnica	Članki	Srednje	0.10.1
javancss	30565	Knjižnica	Orodje	Srednje	33.54
jdom	50384	Knjižnica	Članki	Srednje	2.0.0
jenkins	135962	Knjižnica	Najdeno	Visoko	2.36
jfreechart	140671	GUI aplikacija	Članki	Visoko	1.0.19
joda-time	85847	Knjižnica	Najdeno	Srednje	2.9.6
jsoniter	3686	Knjižnica	Najdeno	Nizko	0.9.3
junit	27820	Knjižnica	Članki	Srednje	4.12
libGDX	259042	Knjižnica	Najdeno	Visoko	1.9.5
log4j	30287	Knjižnica	Najdeno	Srednje	1.2.17
neo4j	618216	Knjižnica	Najdeno	Visoko	3.0.8
pentaho	216013	GUI aplikacija	Najdeno	Visoko	6.1.0.9
slf4j	12483	Knjižnica	Najdeno	Srednje	1.7.22
sonar-qube	575174	Knjižnica	Orodje	Visoko	6.2
sonar-scanner	1784	Knjižnica	Orodje	Nizko	6.2
spring	549476	Knjižnica	Članki	Visoko	5.0.0
spring-boot	194534	Knjižnica	Najdeno	Visoko	1.4.2
treeview3	40969	GUI aplikacija	Članki	Srednje	3.0
worldwind	346469	GUI aplikacija	Najdeno	Visoko	2.1.0
wro4j	40490	Knjižnica	Najdeno	Srednje	1.8.0

Priloga F: Opis uporabljenih projektov

- Activiti – https://github.com/Activiti/Activiti

Je odprtokodno orodje, ki omogoča izvedbo poslovnih procesov, ki so bili napisani v notaciji BPMN.

- Alfresco – https://github.com/Alfresco/community-edition

Je eden izmed večjih ponudnikov repozitorijev za podjetja. Omogoča hrambo dokumentov, sodelovanje, zapisov, baze znanja in storitev spletnih vsebin.

- Ant – https://github.com/apache/ant

Je javanska knjižnica, ki omogoča opis procesov za izgradnjo aplikacija in definicijo razširitvenih točk.

- Apache POI – https://github.com/apache/poi

Je knjižnica za manipuliranje z različnimi formati, kot je recimo Office Open XML standard in Microsoftov OLE2. Knjižnica omogoča zapisovanje podatkov v Excel tabele.

- Arduino – https://github.com/arduino/Arduino

Arduino je strojna komponenta, ki omogoča poganjanje preprostih programov. Orodje omogoča zagon teh programov.

- Ckjm -– https://github.com/dspinellis/ckjm

Program omogoča izračun Chidamber in Kemererjevih objektno-orientiranih metrik na podlagi strojno prevedene kode. Metrike so lahko nato uporabljene za izračun vzdrževalnosti.

- Commons-lang – https://github.com/apache/commons-lang

Javanska knjižnica, ki vsebuje določene pogosto uporabljene javanske razrede, ki niso dovolj standardni, da bi bili vsebovani v java.lang paketu.

- Couchbase – https://github.com/couchbase/couchbase-jvm-core

Couchase JVM core modul je knjižnica, ki skrbi za povezavo med Couchbase strežnikom in odjemalcem. Je namenjena razvoju na dokaj nizkem nivoju, zato je izpostavljen sporočilno naravnan API (vmesnik za namensko programiranje).

- Ditaa – https://github.com/stathissideris/ditaa

Ditaa je orodje v obliki ukaznega okna, ki omogoča pretvorbo slik narisanih v ascii obliki (slike iz znakov, kot so |, / in -) v bitmap format.

- Drill – https://github.com/apache/drill

Apache Drill je porazdeljen povpraševalni nivo, ki podpira uporabo SQL in alternativ za izvajanje v NoSQL in Hadoop podatkovni sistemih.

- Easymock – https://github.com/easymock/easymock

EasyMock je javanska knjižnica, ki omogoča ustvarjanje Mock objektov za uporabo v testih enot.

- FreeMind – https://sourceforge.net/projects/freemind/

FreeMind je aplikacija, ki je uporabljena za vodenje znanja in vsebin. Omogoča enostavno hrambo in pregled nad različnimi vsebinami.

- Gephi – https://github.com/gephi/gephi

Gephi je odprtokodna platforma za vizualizacijo in manipulacijo z večjimi grafi ter podpira različne operacijske sisteme in jezike. Prav tako omogoča izris množice z zelo velikim številom podatkov.

- Gifencode – https://github.com/square/gifencoder

Gifencode je preprosta javanska knjižnica, ki implementira GIF89a specifikacijo in je lahko uporabljena za prikaz slik na Android in ostalih platformah, ki ne uporabljajo razredov iz knjižnice AWT (Abstract Window Toolkit).

- Giraph – https://github.com/apache/giraph

Giraph je orodje za izris grafov, ki so ustvarjani na podlagi akcije v Hadoop infrastrukturi ter omogoča izris grafov iz podatkov, ki jih imamo shranjenih v Hadoop. Prav tako orodje omogoča hkratno procesiranje več podatkovnih baz naenkrat, kar pomeni, da je orodje skalabilno.

- Guava – https://github.com/google/guava

Guava je zbirka knjižnic, ki vključuje razne nove oblike podatkovnih zbirk, kot sta multimap in multiset, nespremenljivi seznam, razne grafe ter različna ostala orodja, ki so lahko priročna pri razvoju aplikacij.

- Hadoop – https://github.com/apache/hadoop

Appache Hadoop je ogrodje, ki omogoča hkratno porazdeljeno procesiranje večjih količin podatkov na različnih klusterjih z uporabo preprostih programerskih modelov. Orodje je pripravljeno predvsem na skalabilnost, tako da ga lahko uporabljamo na enem ali več tisoč strežnikih.

- Hibernate – https://github.com/hibernate/hibernate-orm

Hibernate ORM je ogrodje, ki omogoča objektno-relacijsko mapiranje objektov v programskem jeziku ter njihovo avtomatsko shranjevanje v podatkovno bazo. Prav tako je omogočena implementacija JPA, ki je standardna javanska knjižnica za mapiranje objektov.

- Inbloom – https://github.com/inbloom/java-sdk

Inbloom je javanska knjižnica, ki je namenjena lažjemu razvoju kompatibilnih aplikacij, ki jih med seboj povežemo z zavarovano API komunikacijo.

- Ip-scan – https://github.com/angryziber/ipscan

Angry IP Scanner je skener za preverjanje omrežja, natančneje zasedenih IP naslovov in vrat v omrežju. Uporabljen je predvsem s strani omrežnih administratorjev za vodenje omrežja.

- Javageom – https://github.com/dlegland/javaGeom

JavaGeom je knjižnica, ki ponuja metode za lažje izvajanje geometrijskih kalkulacij, kot je recimo najdba točke, kjer se sekata dve premici oziroma različni geometrijski objekti.

- JavaNCSS – https://github.com/codehaus/javancss

JavaNCSS je preprosto ukazno orodje, ki omogoča izračun internih metrik iz izvorne kode javanskih aplikacij. Metrike so lahko izračunane globalno, glede na razred oziroma za vsako funkcijo.

- JDOM – https://github.com/hunterhacker/jdom

JDOM je orodje, ki omogoča uporabo zapisa XML v Javi. Orodje omogoča pretvorbo seznamov in ostalih kompleksnejših podatkovnih modelov, ki jih standardna Java ne omogoča, v XML elemente. Orodje je optimizirano za uporabo v Javi.

- Jenkins – https://github.com/jenkinsci/jenkins

Jenkins je najpogostejše uporabljen strežnik, ki omogoča avtomatizacijo izvajanj in vodenje celovitega razvoja (continuous integration – ci). Za sam strežnik obstaja veliko število vtičnikov, ki omogočajo avtomatsko izgradnjo in prenos projektov na strežnik.

- JFreeChart – https://github.com/jfree/jfreechart

JFreeChart je orodje, ki omogoča vključevanje grafov v javanske aplikacije, ki je lahko uporabljano tako na strežniški strani kakor tudi na strani odjemalca ter podpira prenos v različnih formatih, kot so SVG, PNG in PDF.

- Joda-time – https://github.com/JodaOrg/joda-time

Joda-Time je javanska knjižnica, ki predstavlja izboljšano verzijo standardnih javanskih knjižnic za delo z datumi in časom. Omogoča različne časovne formate, manipuliranje z datumi ter ostale funkcionalnosti.

- Jsoniter – https://github.com/json-iterator/java

Jsoninter je hiter in fleksibilen razčlenjevalnik za zapis JSON v programerskima jezikoma Java in Go. Omogoča različne načine razčlenjevanja ter je eno izmed najhitrejših orodij, ki so trenutno na voljo.

- JUnit – https://github.com/junit-team/junit4

JUnit je preprosto ogrodje, ki omogoča zapis ponovljivih testov. Kot lahko sklepamo že iz imena, se uporablja za testiranje posameznih enot.

- libGDX – https://github.com/libgdx/libgdx

libGDX je več-platformno javansko ogrodje za razvoj iger, ki temelji na OpenGL in je tako možna njegova uporaba na različnih operacijskih sistemih in brskalnikih, ki podpirajo OpenGL.

- Log4j – https://github.com/apache/log4j

Log4j je ena izmed najbolj pogosto uporabljenih knjižnic za beleženje in hranjenje informacij o delovanju aplikacij. Omogoča beleženje informacij glede na njihovo pomembnosti oziroma različne ostale konfiguracije.

- Neo4j – https://github.com/neo4j/neo4j

Neo4j je vodilna baza za hranjenje grafov. Omogoča shranjevanje vseh potrebnih informacij za hrambo teh in različne tipe transakcij za manipulacijo z njimi.

- Pentaho – https://github.com/pentaho/pentaho-platform

Pentaho je orodje za Business Inteligence, ki omogoča analizo velikih količin podatkov ter njihovo predstavo v obliki, ki je razumljiva za ljudi. Orodje ima več integracijskih točk, raznih OLAP storitev in ostalih možnosti, ki omogočajo integracijo orodja z drugimi.

- Slf4j – https://github.com/qos-ch/slf4j

SLF4J je naslednik orodja log4j, ki je zaradi svoje odvisnosti od starejših verzij Jave težko vzdržljiv. SLF4J omogoča lažje pisanje dnevnikov, pri čemer se lahko uporabnik odloči, katero izmed knjižnic bo uporabil.

- Sonar-qube – https://github.com/SonarSource/sonarqube

SonarQube je platforma, ki omogoča avtomatsko analizo kode in pregled nad kakovostjo kode skozi čas in ali se vzdrževalnost določene programske opreme izboljšuje ali slabša.

- Sonar-scanner – https://github.com/SonarSource/sonar-scanner-cli

Sonar-scanner je orodje, ki omogoča zagon analize v orodju SonarQube. Za zagon je potrebna konfiguracijska datoteka, kjer navedemo vse potrebne informacije. Orodje zaganjamo iz ukazne vrstice.

- Spring – https://github.com/spring-projects/spring-framework

Spring ogrodje predstavlja celotno potrebno infrastrukturo in konfiguracijski model za moderno poslovno aplikacijo, ki temelji na Javi. Prednost Spring je predvsem v tem, da ne temelji na nobenem od razvojnih okolij ter ga lahko tako uporabimo bilo kje.

- Spring Boot – https://github.com/spring-projects/spring-boot

Spring Boot je orodje, ki omogoča lažji zagon aplikacij, ki temeljijo na Spring ogrodju. Spring je lahko tako uporabljen za zagon preprostih JAR aplikacij, kakor tudi kompleksnejših WAR aplikacij. Prav tako vključeje uporabniški vmesnik, ki omogoča izvajanje iz ukazne vrstice.

- Treeview – https://sourceforge.net/projects/jtreeview/

Java Treeview je odprto-kodno orodje, ki omogoča pregled Microarray podatkov, ki so v PCL ali CDT formatu.

- Worldwind – https://github.com/NASAWorldWind/WorldWindJava

Worldwind je javanska aplikacija, ki je razvita s strani ameriške spletne agencije Nasa. V aplikaciji je omogočen pregled zemlje v 3D formatu in gibanje po terenu. Razvijalci lahko dodajo lastne teksture in različne geometrijske objekte v aplikacijo.

- Wro4j – https://github.com/wro4j/wro4j

Wro4j je orodje za analizo in optimizacijo spletnih virov. Združuje več trenutno pogosto uporabljenih orodij, kot so JsMin, JsHint, Css Variables Suport, Less in podobne knjižnice. Wro4j tako omogoča optimizacijo vključitve knjižnic.

Priloga G: Rezultati F-testa

	MI	*MIv2*
Mean	56,73563	57,19837
Variance	134,0248	130,5599
Observations	40	40
df	39	39
F	1,026539	
P(F<=f) one-tail	0,467616	
F Critical one-tail	1,704465	

	MI	*MIwLOC*
Mean	56,73563	54,31172
Variance	134,0248	141,9903
Observations	40	40
df	39	39
F	0,943901	
P(F<=f) one-tail	0,428926	
F Critical one-tail	0,586694	

	MI	*MIvMartins*
Mean	56,73563	46,91585
Variance	134,0248	32,45227
Observations	40	40
df	39	39
F	4,129906	
P(F<=f) one-tail	1,14E-05	
F Critical one-tail	1,704465	

	MI	*MIvCK*
Mean	56,73563	45,32578
Variance	134,0248	9,910704
Observations	40	40
df	39	39
F	13,52324	
P(F<=f) one-tail	2,18E-13	
F Critical one-tail	1,704465	