УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

РЕШЕЊЕ ЗА РУЧНО АНОТИРАЊЕ МИРИСА КОДА ЗАСНОВАНО НА ПРЕСКРИПТИВНОЈ АНОТАЦИОНОЈ ПАРАДИГМИ

ДОКТОРСКА ДИСЕРТАЦИЈА

Ментор:
др Никола Лубурић

Кандидат:
Симона Прокић

Нови Сад, 2025.

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

ОБРАЗАЦ – 5а

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА[1]

Врста рада:	Докторска дисертација
Име и презиме аутора:	Симона Прокић
Ментор: (титула, име, презиме, звање, институција)	др Никола Лубурић, доцент, Факултет техничких наука, Универзитет у Новом Саду
Наслов рада:	Решење за ручно анотирање мириса кода засновано на прескриптивној анотационој парадигми
Језик публикације (писмо):	Српски (латиница)
Физички опис рада:	Унети број Страница: 101 Поглавља: 7 Референци: 124 Табела: 33 Слика: 28 Листинга: 6 Графикона: 0 Прилога: 0
Научна област:	Електротехничко и рачунарско инжењерство
Ужа научна област (научна дисциплина):	Примењене рачунарске науке и информатика
Кључне речи / предметна одредница:	мирис кода, ручно анотирање, одрживост софтвера, квалитет софтвера
Резиме на језику рада:	Мириси кода су структуре у програмском коду које могу нарушити одрживост софтвера. Ручно анотирање скупова података за мирисе кода је изазовно, те је обучавање модела машинског учења (модела МУ) за препознавање мириса отежано. Квалитет скупова података је нарушен неконзистентним анотацијама, малим бројем анотација, нереалистичним односом мириса кода и чистог кода и ограниченом покривеношћу мириса. У овом истраживању је пројектовано решење за ручно анотирање мириса кода засновано на прескриптивној анотационој парадигми. Решење адресира изазове који се јављају приликом анотирања мириса, тј., скупова података за обучавање модела МУ за препознавање мириса. Решење се састоји од анотационе процедуре и алата за анотирање. Валидација решења је спроведена кроз експеримент у ком су три анотатора анотирала мирисе кода. Анотирано је пет мириса кода: дугачка метода (ДМ), велика класа (ВК), класа података (КП), завист међу одликама (ЗМО) и одбачено наследство (ОН). Доприноси истраживања су (1) решење за ручно анотирање мириса кода засновано на прескриптивној анотационој парадигми, (2) анотациона шема и смернице за ДМ, ВК, КП, ЗМО и ОН, (3) ручно анотирани скуп података за ДМ, ВК, КП, ЗМО и ОН, (4) алат за анотирање мириса кода.

[1] Аутор докторске дисертације потписао је и приложио следеће Обрасце:
5б – Изјава о ауторству;
5в – Изјава о истоветности штампане и електронске верзије и о личним подацима;
5г – Изјава о коришћењу.
Ове Изјаве се чувају на факултету у штампаном и електронском облику и не кориче се са тезом.

	Ови доприноси су значајни за следеће интересне групе: анотаторе који креирају скупове података високог квалитета за обучавање модела МУ, истраживаче из области МУ који креирају моделе за препознавање мириса кода и софтверске инжењере који користе моделе МУ за повећање одрживости софтвера.
Датум прихватања теме од стране надлежног већа:	29.1.2025.
Датум одбране: (Попуњава одговарајућа служба)	
Чланови комисије: (титула, име, презиме, звање, институција)	Председник: др Игор Дејановић, редовни професор, Факултет техничких наука, Универзитет у Новом Саду Члан: др Јелена Сливка, ванредни професор, Факултет техничких наука, Универзитет у Новом Саду Члан: др Силвиа Гилезан, редовни професор, Факултет техничких наука, Универзитет у Новом Саду Члан: др Душан Савић, ванредни професор, Факултет организационих наука, Универзитет у Београду Члан: др Никола Лубурић, доцент, Факултет техничких наука, Универзитет у Новом Саду
Напомена:	

UNIVERSITY OF NOVI SAD
FACULTY OF TECHNICAL SCIENCES

KEY WORD DOCUMENTATION[2]

Document type:	Doctoral dissertation
Author:	Simona Prokić
Supervisor (title, first name, last name, position, institution):	Nikola Luburić, Ph.D., Assistant Professor, Faculty of Technical Sciences, University of Novi Sad
Thesis title:	A solution for manual code smell annotation based on prescriptive annotation paradigm
Language of text (script):	Serbian language (latin script)
Physical description:	Number of: Pages: 101 Chapters: 7 References: 124 Tables: 33 Illustrations: 28 Listings: 6 Graphs: 0 Appendices: 0
Scientific field:	Electrical and computer engineering
Scientific subfield (scientific discipline):	Applied computer science and informatics
Subject, Key words:	code smell, manual annotation, software maintainability, software quality
Abstract in English language:	Code smells are structures in code that present potential software maintainability issues. Manually constructing high-quality datasets to train ML models for code smell detection is challenging. Inconsistent annotations, small dataset size, non-realistic smell-to-non-smell ratio, and poor smell coverage hinder the dataset quality. To address challenges related to building high-quality datasets suitable for training ML models for smell detection, we designed a solution for manual code smell annotation based on prescriptive annotation paradigm. A solution consists of annotation procedure and annotation tool. We validated the solution by employing three annotators to annotate smells following the proposed solution. Three annotators annotated five code smells: Long Method (LM), Large Class (LC), Data Class (DC), Feature Envy (FE), and Refused Bequest (RB). Contributions of this research are (1) solution for manual code smell annotation based on prescriptive annotation paradigm, (2) annotation schema and guidelines for LM, LC, DC, FE and RB, (3) manually annotated dataset for LM, LC, DC, FE and RB, (4) annotation tool. These contributions benefit several stakeholders: annotators building high-quality smell datasets that can be used to train ML models, ML researchers building ML models for smell detection, and software engineers employing ML models to enhance the software maintainability.
Accepted on Scientific Board on:	29.1.2025.

[2] The author of doctoral dissertation has signed the following Statements:
5б – Statement on the authority,
5в – Statement that the printed and e-version of doctoral dissertation are identical and about personal data,
5г – Statement on *copyright licenses.*
The paper and e-versions of Statements are held at he faculty *and are not included into the printed thesis*.

Defended: (Filled by the faculty service)	
Thesis Defend Board: (title, first name, last name, position, institution)	Chair: Igor Dejanović, Ph.D., Full Professor, Faculty of Technical Sciences, University of Novi Sad Member: Jelena Slivka, Ph.D., Associate Professor, Faculty of Technical Sciences, University of Novi Sad Member: Silvia Gilezan, Ph.D., Full Professor, Faculty of Technical Sciences, University of Novi Sad Member: Dušan Savić, Ph.D., Associate Professor, Faculty of Organizational Sciences, University of Belgrade Member: Nikola Luburić, Ph.D., Assistant Professor, Faculty of Technical Sciences, University of Novi Sad
Note:	

Zahvalnica

Želim da izrazim iskrenu zahvalnost mentoru, dr Nikoli Luburiću, na podršci, nesebičnim savetima i prenetom znanju. Njegova posvećenost i trud su me oduvek inspirisali.

Zahvaljujem se i profesorima i kolegama koji su svojim sugestijama doprineli poboljšanju kvaliteta ovog istraživanja.

Najveću zahvalnost dugujem porodici koja je uvek bila tu za mene. Za kraj, zahvaljujem se Deji, bez čije podrške bi sve bilo mnogo teže.

Sadržaj

SPISAK KORIŠĆENIH SKRAĆENICA

SKRAĆENICA	PUN NAZIV NA ENGLESKOM
SDLC	Software Development Life Cycle
ML	Machine Learning
NLP	Natural Language Processing
SRP	Single Responsibility Principle
WMC	Weighted Method Count
WMC_NO_CASE	Weighted Method Count that does not count case
AST	Abstract Syntax Tree
CBO	Coupling Between Objects
DIT	Depth of Inheritance Tree
IAA	Inter-Annotator Agreement
NOM (NMD)	Number Of Methods
NOA (NAD)	Number Of Attributes
POC	Proof-Of-Concept
DSE	DataSet Explorer
LOC	Lines Of Code
CYCLO (VG)	Cyclomatic Complexity
NOF	Number Of Fields
NOAM	Number Of Accessor Methods
NOPA	Number Of Public Attributes
WOC	Weight Of Class
LCOM, LCOM3, LCOM4	Lack of Cohesion between Methods
xMC	External Method Call
AID	Access of Import Data
ATFD, ATFD_10	Access To Foreign Data
IMC	Internal Method Call
ALD	Access of Local Data
BUR	Base-class Usage Ratio
BOvR	Base-class Overriding Ratio
NOPP	Number Of Public Properties
MANOVA	Multivariate Analysis of Variance
CLOC	Lines Of Code of a Class
MLOC	Lines Of Code in a Method
NIC	Number of Inner Classes
C3	Conceptual Cohesion of Classes
ANOVA	Analysis of Variance
NMD_NAD	Number of Methods Declared and Number of Attributes Defined
CYCLO_SWITCH	Cyclomatic Complexity that does not count every case label, only the Switch label
MELOC	Effective Lines Of Code in a Method
NOP	Number of Parameters
NOLV	Number Of Local Variables
NOTC	Number Of Try Catch blocks
MNOL	Number Of Loops in a Method
NOR	Number Of Return statements in a method
MNOC	Number Of Comparison operators in a Method
MNOA	Number Of Assignments in a Method
NONL	Number Of Numeric Literals
NOSL	Number Of String Literals
NOMO	Number Of Math Operations
NOPE	Number Of Parenthesized Expressions
NOLE	Number Of Lambda Expressions
MMNB	Max Nested Blocks in a Method

NOUW	Number Of Unique Words
CELOC	Effective Lines Of Code of a Class
TCC	Tight Class Cohesion
CNOR	Number Of Return statements from all Class members
CNOL	Number Of Loops from all Class members
CNOC	Number Of Comparison Operators from all Class members
CNOA	Number Of Assignments from all Class members
NOPM	Number Of Private Methods
NOPF	Number Of Protected Fields
DCC	Direct Class Coupling
CMNB	Max Nested Blocks from all Class members
RFC	Response For a Class, counting the number of unique method invocations

SPISAK SLIKA

SPISAK TABELA

SPISAK LISTINGA

1. UVOD

Održavanje softvera je ključno za ispravan rad softvera i prilagođavanje izmenama u zahtevima korisnika i okruženja. Troškovi održavanja su manji ukoliko je softver održiv, jer ga je lakše popravljati, menjati i unapređivati. Održivost softvera mogu narušiti mirisi koda koji otežavaju razumevanje, čitljivost i izmenu softvera. Štetne mirise koda koji ukazuju na ozbiljnije probleme u softveru bi trebalo otkriti i ukloniti, za šta se mogu koristiti modeli mašinskog učenja.

Performanse modela mašinskog učenja zavise od kvaliteta skupova podataka na kojima se obučavaju. Ručno anotirani skupovi su potencijalno najkvalitetniji za obučavanje. Međutim, kvalitet je često narušen nekonzistentnim anotacijama, malim brojem anotacija, nerealističnim odnosom mirisa koda i čistog koda i ograničenom pokrivenošću mirisa. Ovo istraživanje se fokusira na projektovanje rešenja za ručno anotiranje mirisa koda koje bi omogućilo kreiranje kvalitetnih skupova podataka. Time bi se doprinelo efikasnijoj detekciji mirisa, a zatim i povećanju održivosti softvera i smanjenju troškova održavanja softvera.

1.1 Kontekst problema

Održivost koda je jedan *aspekt kvaliteta* softvera koji se odnosi na lakoću kojom se softver menja u cilju ispravke grešaka, unapređenja performansi ili prilagođavanja softvera novom okruženju [1]. Međunarodni standard *ISO 25010* definiše model kvaliteta softvera koji služi za evaluaciju kvaliteta softverskog proizvoda [2]. Model definiše karakteristike i podkarakteristike kvaliteta koje treba uzeti u obzir prilikom evaluacije. Jedna od karakteristika je *održivost* koja je definisana kao efikasnost i efektivnost kojom se softver menja radi unapređenja, popravke ili prilagođavanja promenama u zahtevima i okruženju [2]. Po ovom standardu, održivost se može dekomponovati na sledeće podkarakteristike:

- Modularnost (engl. *modularity*), koja predstavlja stepen do kog je softver podeljen na nezavisne komponente tako da promene jedne komponente imaju što manji uticaj na druge komponente.
- Mogućnost ponovne upotrebe (engl. *reusability*), koja predstavlja stepen do kog se softver ili komponente softvera mogu iskoristiti u drugim kontekstima.
- Mogućnost analize (engl. *analysability*), definisana kao efikasnost i efektivnost kojom se (1) procenjuje uticaj izmena jedne komponente na ostatak softvera, (2) identifikuju nedostaci ili greške, (3) identifikuju komponente koje treba menjati.
- Mogućnost izmene (engl. *modifiability*), definisana kao efikasnost i efektivnost izmene softvera bez uvođenja grešaka ili smanjenja kvaliteta softvera.
- Mogućnost testiranja (engl. *testability*), predstavljena kao efikasnost i efektivnost kreiranja kriterijuma koje softver treba da ispuni, te definisanja testova koji utvrđuju ispunjenost kriterijuma.

Karakteristike i podkarakteristike kvaliteta softvera definisane *ISO 25010* standardom su predstavljene na Slici 1.1.

Slika 1.1 Međunarodni standard *ISO 25010* [2] definiše 9 karakteristika kvaliteta softvera. Svaka karakteristika je dekomponovana na podkarakteristike koje preciznije definišu aspekte kvaliteta softvera. Ovo istraživanje se bavi aspektom održivosti softvera, koje je naglašeno na slici.

Životni ciklus razvoja softvera (engl. *Software Development Life Cycle* - SDLC) je proces koji ima za cilj razvijanje kvalitetnog softvera. Sprovođenjem procesa se proizvodi softver koji je pušten u rad, gde služi krajnje korisnike. Nakon puštanja u rad, softver je potrebno održavati kako bi se prilagodio novim potrebama korisnika i ostao relevantan u okruženju koje se menja. *Održavanje* je ključna faza u životnom ciklusu razvoja softvera i podrazumeva ispravku grešaka, poboljšanje performansi, prilagođavanje promenama u okruženju i usklađivanje sa novim ili izmenjenim zahtevima korisnika [3][4]. Održavanje može biti najskuplja faza SDLC-a, dostižući i do 80% ukupne cene razvoja softvera [3][4][5].

Jedan od glavnih razloga za visoku cenu održavanja softvera je niska održivost njegovog koda. Niska održivost se manifestuje kroz nejasne nazive identifikatora u kodu, dugačke ili kompleksne funkcije koje se teško analiziraju, menjaju i testiraju, kao i module koji su spregnuti sa mnoštvom koda tako da ih je teško koristiti na više mesta [3]. Zbog niske održivosti koda, programeri koji održavaju softver mogu potrošiti i do 60% svog vremena na razumevanje koda [5]. Povećanjem održivosti koda se povećava efikasnost održavanja softvera i smanjuje ukupan troška razvoja softvera. Posledično će takav kod moći brže da se prilagodi izmenama na tržištu, što povećava uspešnost kompanije koja naručuje softver, kao i one koja ga proizvodi [6][7].

Mirisi koda (engl. *code smells*) predstavljaju strukture u programskom kodu koje ukazuju na moguće probleme u softveru koji narušavaju održivost koda [8]. Termin *code smell* je prvi put upotrebio Kent Beck u diskusiji sa Martinom Fowlerom, opisujući kod čijim refaktorisanjem[3] bi se povećala održivost softvera [8]. U knjizi [8] su definisana 22 mirisa koda. Iako mirisi koda ne predstavljaju greške i ne sprečavaju ispravan rad softvera, oni ukazuju na kod koji je teško razumeti, prilagoditi, unaprediti ili ispraviti, što otežava održavanje softvera [9][10].

[3] Refaktorisanje predstavlja promenu softvera koja treba da olakša njegovo razumevanje i smanji buduće troškove menjanja. Refaktorisanje se vrši primenom niza aktivnosti koje ne menjaju ponašanje, već samo internu strukturu softvera.

Stoga se inženjeri softvera slažu da štetne[4] mirise koda treba otkriti i ukloniti radi poboljšanja kvaliteta koda [9][11].

Tradicionalan pristup automatskoj detekciji mirisa koda je razvoj ručno definisanih pravila koja porede strukturne metrike[5] koda sa predefinisanim pragovima [12]. Međutim, istraživačka zajednica nije usaglašena po pitanju skupa strukturnih metrika koje treba razmatrati pri detekciji određenog mirisa koda, kao ni po pitanju pragova koje treba postaviti za određene metrike [12][13][14]. Ovaj problem bi mogao biti prevaziđen treniranjem modela mašinskog učenja (engl. *Machine Learning – ML*) koji su u mogućnosti da nauče pravila na osnovu anotiranih[6] primera [12][15].

Performanse ML modela zavise od kvaliteta skupa podataka korišćenog za njihovo treniranje. Postoje tri glavna pristupa anotaciji skupa podataka: automatska anotacija, poluautomatska anotacija i ručna anotacija. Skupovi podataka se mogu anotirati automatski primenom ručno definisanih pravila. Međutim, ovakvi skupovi podataka nisu adekvatni za treniranje ML modela, budući da bi rezultujući ML modeli predstavljali nesavršenu imitaciju primenjenog pravila, koje pravi greške pri anotaciji [9][16]. Poluautomatska anotacija skupa podataka ima za cilj da smanji ljudski trud pri anotaciji tako što primenjuje ručno definisana pravila radi generisanja skupa kandidata koji predstavljaju potencijalne mirise koda. Proizvedene kandidate pregledaju ljudski stručnjaci u cilju uklanjanja lažno pozitivnih anotacija. Međutim, ova postavka ne garantuje da skup podataka ne sadrži lažno negativno anotirane instance, odnosno, instance koje su primenjena pravila pogrešno eliminisala [15]. Zato je za kreiranje kvalitetnog skupa podataka, neophodnog za obuku ML modela visokih performansi, potrebno da skup podataka bude u potpunosti ručno anotiran [16][17].

Visok kvalitet skupova podataka je značajan za nekoliko interesnih grupa. *Anotatori* imaju cilj da izgrade skupove podataka visokog kvaliteta koji se mogu koristiti za obučavanje ML modela za detekciju mirisa koda. Zatim, *ML istraživači* nastoje da stvore robusne i nepristrasne ML modele za detekciju mirisa koda. Važnost korišćenja skupova visokog kvaliteta za obučavanje i poređenje ML modela je istaknuta u više prethodnih istraživanja [18][19][20]. Na kraju, *softverski inženjeri* teže poboljšanju održivosti koda, jer povećana održivost olakšava rad i smanjuje troškove razvoja softvera. ML modeli obučeni na skupovima podataka visokog kvaliteta bi bili korisni za poboljšanje održivosti softvera [12][15].

Ručno anotiranje mirisa koda je izazovan zadatak. Definicije mirisa su zasnovane na heuristikama koje su podložne subjektivnoj interpretaciji. Ove heuristike mogu zavisiti od programskog jezika, radnog okvira i konvencija. Rezultati primene heuristika u praksi zavise od iskustva i intuicije stručnjaka koji ih primenjuju [21], što se odražava u značajnim nesuglasicama među stručnjacima u primeni definicija mirisa koda na konkretan kod [15][22]. Nesuglasice bi mogle biti smanjene preciznijim smernicama za prepoznavanje mirisa koda i

[4] Mirisi koda su pokazatelji mogućih problema koji mogu, ali ne moraju da značajno redukuju održivost softvera. Štetni mirisi koda ukazuju na sigurne probleme koji ozbiljnije narušavaju održivost koda i potrebno ih je ukloniti.

[5] Primeri strukturnih metrika bi bili „broj linija koda“ i „broj atributa u klasi“.

[6] Pod *anotiranjem* se podrazumevaju sledeće aktivnosti: analiza različitih svojstava koda (na primer, izvorni programski kod i strukturne metrike), detekcija mirisa koda, određivanje njegovog intenziteta i formiranje anotacije.

primerima za prepoznavanje mirisa koda, ali postoji veoma malo pokušaja da se one razviju [23]. Još jedan problem je što anotiranje mirisa koda zahteva detaljnu analizu semantike koda, što je vremenski zahtevan proces. Stoga su postojeći skupovi podataka često mali [12][15].

Nekonzistentne anotacije i mali broj anotacija su neki od fenomena koji umanjuju kvalitet ručno anotiranih skupova podataka. Problem nekvalitetnih skupova podataka negativno utiče na performanse ML modela obučavanih za detekciju mirisa koda, te oni nisu korisni za poboljšanje održivosti softvera.

1.2 Opis problema

Za razumevanje problema nekvalitetnih skupova podataka je neophodno sagledati fenomene koji doprinose lošem kvalitetu skupova u kontekstu mirisa koda, kao i njihove uzroke i uticaj. U nastavku su opisana četiri identifikovana fenomena:

1. Nekonzistentne anotacije,
2. Mali broj anotacija,
3. Nerealističan odnos mirisa koda i čistog koda,
4. Ograničena pokrivenost mirisa koda.

Nekonzistentne anotacije mogu nastati zbog nenamernih grešaka prilikom anotiranja koje su posledica umora ili ometene pažnje anotatora. Greške podrazumevaju previde pri analizi mirisa koda ili pogrešno zabeležene anotacije. Verovatnoća greške se povećava ako je isključivo jedan anotator zadužen za anotiranje skupa podataka jer ne postoji mehanizam za proveru anotacija [15]. Drugi razlog nastanka nekonzistentnih anotacija je odsustvo preciznih smernica za anotiranje. Budući da su definicije mirisa neprecizne heuristike podložne subjektivnoj interpretaciji [22], anotatori različitih nivoa znanja i iskustva nisu usaglašeni po pitanju svojstava koda koje treba analizirati kako bi se odredilo da li on predstavlja određeni mirisa koda [21]. Posledično, anotacije istog koda dobijene od različitih anotatora mogu biti neusaglašene. Ovaj problem bi mogao biti umanjen definisanjem preciznih smernica za anotiranje koje bi obezbedile zajednički radni okvir za sve anotatore [23].

Fenomen nekonzistentnih anotacija se može pronaći i u oblasti obrade prirodnog jezika (engl. *Natural Language Processing* - NLP). Anotiranje skupova podataka za NLP algoritme podrazumeva semantičku analizu teksta, gde se anotatori susreću sa dvosmislenim i nepreciznim podacima koje subjektivno tumače. Ta subjektivnost može rezultirati neslaganjima koja umanjuju kvalitet skupa podataka [23]. Zbog ove sličnosti, anotacione prakse iz NLP oblasti se mogu primeniti pri anotiranju mirisa koda.

Mali broj anotacija je posledica vremenske zahtevnosti ručnog anotiranja. Anotiranje mirisa koda podrazumeva razmatranje više svojstava analiziranog koda i dublje razumevanje celokupnog softverskog sistema u kome se on nalazi [12]. Anotatori za različite mirise koda moraju da ispitaju različita svojstva koda, što znači da anotiranje svakog pojedinačnog mirisa koda podrazumeva zasebnu analizu.

Drugi faktor koji uzrokuje ograničenje veličine skupova podataka je potreba za stručnjacima koji mogu vršiti ovaj tip analize. Anotiranje mirisa koda zahteva da anotatori imaju specifičnu ekspertizu kako bi mogli razumeti dizajn i semantiku koda [15]. Anotatori bi trebalo da poseduju znanje o razvoju i arhitekturi softvera, principima dizajna, programskim jezicima,

mirisima koda i tehnikama refaktorisanja.

Nerealističan odnos mirisa koda i čistog koda (engl. *smell-to-non-smell*) u skupu podataka nastaje kada anotatori ciljaju da kreiraju skupove podataka sa izbalansiranim klasama [13]. Međutim, ovakvi skupovi podataka ne oslikavaju realan odnos mirisa koda i čistog koda u softverskim sistemima, jer je u stvarnosti mali deo softvera pod uticajem mirisa koda [24]. Odstupanje od realne distribucije klasa u softverskim sistemima može nepovoljno uticati na generalizaciju rezultujućih ML modela [25].

Iako je Fowler definisao 22 mirisa [8], u skupovima podataka se može primetiti *ograničena pokrivenost mirisa koda*. Zbog vremenske zahtevnosti ručnog anotiranja, anotatori često daju prioritet mirisima koda koje je lakše anotirati [17]. Zakeri-Nasrabadi i saradnici su analizirali skupove podataka u sistematičnom pregledu literature [17] i ukazali na nedostatak istraživanja vezanih za veliki broj mirisa. Od 10 javno dostupnih ručno anotiranih skupova podataka, jedan skup je uključivao tri od 22 mirisa koda definisana od strane Martina Fowlera [26], dva skupa su uključivala četiri mirisa [27][28], dva skupa po pet mirisa [29][30], a samo jedan skup podataka je imao 10 mirisa [31]. Preostali skupovi podataka su sadržali samo jedan od Fowlerovih mirisa koda [32][33][34][35].

1.3 Hipoteze i ciljevi istraživanja

Cilj ovog istraživanja je unapređenje kvaliteta ručno anotiranih skupova podataka vezanih za mirise koda. Postizanje ovog cilja može da doprinese kreiranju i obučavanju ML modela za detekciju mirisa, a zatim i poboljšanju održivosti koda. Povećana održivost koda dovodi do unapređenja kvaliteta softvera i smanjenja troškova razvoja softvera.

Očekivani rezultat ovog istraživanja je rešenje za ručno anotiranje mirisa koda zasnovano na *preskriptivnoj anotacionoj paradigmi*. Preskriptivna anotaciona paradigma se koristi u NLP oblasti za kreiranje kvalitetnih skupova podataka za obučavanje ML modela i poređenje različitih ML pristupa [36][37]. Rešenje treba da obuhvata anotacionu proceduru koja usvaja preskriptivnu paradigmu i alat koji anotatori mogu koristiti za anotiranje skupa podataka.

Hipoteza ovog istraživanja je da rešenje za ručno anotiranje mirisa koda zasnovano na preskriptivnoj anotacionoj paradigmi doprinosi većem kvalitetu skupova podataka u poređenju sa postojećim rešenjima. Hipoteza se može razložiti na podhipoteze, koje omogućavaju precizniju procenu da li rešenje povećava kvalitet skupa:

1. H1: Rešenje će povećati konzistentnost anotacija.
2. H2: Rešenje će ubrzati anotiranje i omogućiti kreiranje većih skupova podataka.
3. H3: Rešenje će omogućiti kreiranje skupova podataka koji oslikavaju realističan odnos koda koji sadrži mirise i koda koji ih ne sadrži.
4. H4: Rešenje će omogućiti kreiranje skupova podataka za sve vrste mirisa koda koje je definisao Martin Fowler.

Na osnovu definisanih ciljeva i hipoteza, rešenje za ručno anotiranje mirisa koda treba da ispuni zahteve navedene u Tabeli 1.1. Zahtevi su grupisani na osnovu fenomena koji negativno utiču na ciljeve interesnih grupa. Pored zahteva, opisani su argumenti doprinosa, koji objašnjavaju kako rešenje koje ispunjava navedene zahteve adresira fenomene i doprinosi ciljevima interesnih grupa. Provera ispunjenosti zahteva će omogućiti proveru ispunjenosti postavljenih hipoteza.

Tabela 1.1 Zahtevi definisani za rešenje za ručno anotiranje mirisa koda i argumenti doprinosa koji opisuju kako ispunjenje zahteva doprinosi ciljevima interesnih grupa.

Fenomen	Zahtev	Argument doprinosa
Nekonzistentne anotacije	Z1: Rešenje treba da uključi obuku anotatora.	Obuka anotatora doprinosi kvalitetu skupova podataka kroz usklađivanje razumevanja anotatora o zadatku anotacije, čime se povećava njihova međusobna saglasnost i konzistentnost anotacija [38][39].
	Z2: Rešenje treba da podrži anotiranje od strane više anotatora.	Učešće više anotatora u procesu anotiranja omogućava validaciju i diskusiju anotacija, što dovodi do pouzdanijih i konzistentnijih anotacija. Iako bi dva anotatora bila dovoljna da se obezbedi validacija i diskusija, uključivanje više anotatora pruža bolju priliku za razmenu ideja i izbegavanje grešaka. Ovakav kolaborativni pristup anotiranju unapređuje kvalitet anotacija jer smanjuje subjektivni uticaj i greške pojedinačnih anotatora [40][41].
	Z3: Rešenje treba da omogući razrešavanje nesuglasica među anotatorima, tj., konfliktnih anotacija.	Razrešavanje nesuglasica i ispravljanje grešaka je neophodno za obezbeđivanje konzistentnih i tačnih anotacija, čime se poboljšava kvalitet skupova podataka [39][42].
Mali broj anotacija	Z4: Rešenje treba da ubrza anotiranje.	Ubrzavanje procesa anotiranja omogućava anotatorima da kreiraju veće skupove podataka.
Nerealističan odnos mirisa koda i čistog koda	Z5: Rešenje treba da omogući kreiranje skupova podataka sa realističnim odnosom mirisa koda i čistog koda.	Odnos mirisa koda i čistog koda treba da oslikava njihov realan odnos u softverskim sistemima. Odnos koji predstavlja stvarnu distribuciju obezbeđuje pouzdaniji skup podataka za obučavanje ML modela za detekciju mirisa koda [25].
Ograničena pokrivenost mirisa koda	Z6: Rešenje treba da omogući anotiranje bilo kog mirisa koda definisanog od strane Martina Fowlera.	Rešenje koji podržava anotiranje bilo kog Fowlerovog mirisa koda omogućava kreiranje obimnih skupova podataka koji sadrže skup različitih vrsta mirisa koda. ML istraživači mogu iskoristiti takve skupove da naprave robusne i pouzdane ML modele za detekciju mirisa.

1.4 Doprinosi disertacije

Doprinosi istraživanja razvrstani u četiri kategorije (naučni, razvojni, aplikativni i socio-ekonomski) su detaljno analizirani u Potpoglavlju 6.1. Doprinosi koji obuhvataju rešenje za ručno anotiranje mirisa koda, anotacione šeme i smernice i anotirani skup podataka su predstavljeni u radovima objavljenim u međunarodnim časopisima i saopštenjima sa međunarodnih skupova:

- Prokić, S., Luburić, N., Slivka, J. and Kovačević, A., Prescriptive Procedure for Manual Code Smell Annotation. *Science of Computer Programming*, 2024**.** p.103168.
- Slivka, J., Luburić, N., Prokić, S., Grujić, K.G., Kovačević, A., Sladić, G. and Vidaković, D., Towards a Systematic Approach to Manual Annotation Of Code Smells. *Science of Computer Programming*, 230 (2023), p.102999.
- Prokić, S., Luburić, N., Slivka, J. and Kovačević, A., Identification of Code Properties That Support Code Smell Analysis. *In 2023 46th MIPRO ICT and Electronics Convention (MIPRO)* 2023. pp. 1664-1669.
- Prokić, S., Grujić, K.G., Luburić, N., Slivka, J., Kovačević, A., Vidaković, D. and Sladić, G., Clean Code and Design Educational Tool, *In 2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*, 2021, pp. 1601-1606.

U okviru istraživanja su objavljeni i radovi u vodećim međunarodnim časopisima koji se bave komplementarnim temama poput analize performansi i poteškoća pri detekciji mirisa koda:

- Kovačević, A., Luburić, N., Slivka, J., Prokić, S., Grujić, K.G., Vidaković, D. and Sladić, G., Automatic Detection of Code Smells Using Metrics and CodeT5

Embeddings: A Case Study in C#. *Neural Computing and Applications*, 36(16) (2024), pp.9203-9220.

- Kovačević, A., Slivka, J., Vidaković, D., Grujić, K.G., Luburić, N., Prokić, S. and Sladić, G., Automatic Detection of Long Method and God Class Code Smells Through Neural Source Code Embeddings. *Expert Systems With Applications*, 204 (2022), p.117607.

1.5 Struktura disertacije

U Poglavlju 2 je izvršen pregled aktuelnog stanja u oblasti kroz opis mirisa koda i analizu detekcije, izazova ručnog anotiranja, postojećih anotacionih procedura i alata za ručno anotiranje mirisa koda. U Poglavlju 3 je opisana metodologija na kojoj je zasnovano ovo istraživanje, a koja uključuje istraživanje problema, projektovanje rešenja i validaciju rešenja. Rešenje za ručno anotiranje mirisa koda je predstavljeno u Poglavlju 4, dok je validacija predstavljena u Poglavlju 5. Doprinosi i ograničenja istraživanja su prodiskutovana u Poglavlju 6, a Poglavlje 7 sadrži zaključak istraživanja.

2. PREGLED AKTUELNOG STANJA U OBLASTI

Predloženo istraživanje se nalazi na preseku dva domena: softverskog inženjerstva (gde analiziramo svojstva koda i mirise koda) i mašinskog učenja (gde ispitujemo prakse i heuristike za anotaciju skupova podataka pogodnih za obuku ML modela).

U Potpoglavlju 2.1 su opisani mirisi koda koje je definisao Martin Fowler, gde su istaknuti primeri najinteresantnijih mirisa za ovo istraživanje u Java programskom jeziku. Pošto je cilj istraživanja kreiranje rešenja za ručno anotiranje mirisa koda, u Potpoglavlju 2.2 su obrađeni izazovi ručnog anotiranja mirisa. Jedan od izazova je odabir relevantnih aspekata, tj., svojstava koda koje treba analizirati prilikom detekcije mirisa, te Potpoglavlje 2.3 predstavlja prethodno istraživanje gde su identifikovana svojstva koda korišćena za detekciju i određena relevantnost svakog svojstva za detekciju 22 Fowlerova mirisa. Sledeći izazov, utvrđivanje strategija, tj., heuristika za detekciju, se javlja zbog nepreciznih i dvosmislenih definicija mirisa. Zbog sličnosti sa anotiranjem mirisa, anotacione prakse iz NLP oblasti su opisane u Potpoglavlju 2.4. Pregled aktuelnog stanja u oblasti je upotpunjen analizom postojećih anotacionih procedura za ručno anotiranje mirisa u Potpoglavlju 2.5, dok su u Potpoglavlju 2.6 analizirani postojeći alati za ručno anotiranje. Za kraj, Potpoglavlje 2.7 rezimira čitavo istraživanje problema u konceptualni radni okvir problema.

2.1 Mirisi koda

Martin Fowler je u svojoj knjizi „Refaktorisanje: Poboljšanje dizajna postojećeg koda“ [8] definisao 22 mirisa koda, koji su navedeni u Tabeli 2.1. Posle njega, mnogi istraživači su predlagali različite definicije istih mirisa koda i definisali nove mirise za različite segmente softverskog rešenja, kao što su arhitektura, testovi i baze podataka [9]. Na osnovu ovih istraživanja, pregledni rad [9] izdvaja sledeće karakteristike koje pomažu u razumevanju mirisa:

- Mirisi koda su indikatori ili simptomi problema u dizajnu softvera.
- Mirisi koda predstavljaju loša rešenja.
- Mirisi koda narušavaju ustanovljene dobre prakse u implementaciji ili dizajnu softvera.
- Mirisi koda otežavaju održavanje softvera, te narušavaju njegov kvalitet.
- Mirisi koda su problemi u kodu koji se iznova pojavljuju.

Istraživači često kategorizuju mirise koda kako bi se olakšala njihova identifikacija i odabir adekvatnih metoda refaktorisanja kojima se uklanjaju. Različita istraživanja su kategorizovala Fowlerove, ali i druge mirise koda [9]. U Tabeli 2.1 su istaknute kategorije koje su dodeljene Fowlerovim mirisima[7]. Kategorije navedene u Tabeli 2.1 će biti opisane u nastavku teksta, dok ih Tabela 2.2 rezimira.

[7] Radovi [43][44] i [45] su kategorizovali manji podskup Fowlerovih mirisa od radova [45][46][47] i [48], te su izostavljeni iz Tabele 2. Od 22 mirisa, [43] i [44] su kategorizovali 14, a [45] samo 9 mirisa.

Tabela 2.1 Spisak Fowlerovih mirisa koda i dodeljenih kategorija. Oznaka „N/A“ za kategoriju označava da određeni miris koda nije kategorizovan u referenciramo radu.

Miris koda	**Definicija**	**Kategorija**	
Ponovljeni kod (engl. *Duplicated Code*)	Isti ili vrlo sličan kod postoji na više mesta u softveru.	• Dispensables [46] • Duplication [47]	• Smells within classes [45] • Dependency-based [48]
Dugačka metoda (engl. *Long Method*)	Metoda je dugačka, kompleksna i ima više odgovornosti.	• Bloaters [46] • Measured smells [47]	• Smells within classes [45] • Mass-based [48]
Velika klasa (engl. *Large Class*)	Klasa ima velik broj polja i metoda i ima više odgovornosti.	• Bloaters [46] • Measured smells [47]	• Smells within classes [45] • Mass-based [48]
Dugačka lista parametara (engl. *Long Parameter List*)	Metoda prima velik broj parametara.	• Bloaters [46] • Measured smells [47]	• Smells within classes [45] • Mass-based [48]
Divergentne izmene (engl. *Divergent Change*)	Klasa se često menja na različite načine iz različitih razloga.	• Change Preventers [46] • Accommodating Change [47]	• Smells between classes [45] • Feature-based [48]
Operacija sačmarom (engl. *Shotgun Surgery*)	Izmena na jednom mestu u softveru zahteva izmene u različitim klasama.	• Change Preventers [46] • Accommodating Change [47]	• Smells between classes [45] • Feature-based [48]
Zavist među odlikama (engl. *Feature Envy*)	Metoda pretežno koristi metode ili atribute druge klase.	• Couplers [46] • Responsibility [47]	• Smells between classes [45] • Dependency-based [48]
Skupine podataka (engl. *Data Clumps*)	Isti ili slični podaci se javljaju zajedno na više mesta u kodu (kao parametri metoda ili polja u klasama).	• Bloaters [46] • Data [47]	• Smells between classes [45] • Structure-based [48]
Opsesija primitivnim podacima (engl. *Primitive Obsession*)	Upotreba primitivnih tipova podataka umesto klasa koje bi enkapsulirale te podatke.	• Bloaters [46] • Data [47]	• Smells between classes [45] • Structure-based [48]
Ponavljanje *switch* naredbe (engl. *Switch Statements*)	Ponavljanje iste *switch* naredbe ili *if-else* logike na više mesta u softveru.	• Object-oriented Abusers [46] • Conditional Logic [47]	• N/A [45] • Dependency-based [48]
Paralelne hijerarhije nasleđivanja (engl. *Parallel Inheritance Hierarchies*)	Dodavanje nove klase u jednoj hijerarhiji zahteva dodavanje nove klase i u drugoj hijerarhiji.	• Object-oriented Abusers [46] • Accommodating Change [47]	• Smells between classes [45] • Dependency-based [48]
Lenja klasa (engl. *Lazy Class*)	Klasa ima malo podataka i metoda, te nema dovoljno značajnu ulogu da bi postojala.	• Dispensables [46] • Inheritance [47]	• Smells between classes [45] • Feature-based [48]
Pretpostavljena uopštenost (engl. *Speculative Generality*)	Delovi koda za koje se pretpostavljalo da će biti potrebni u budućnosti, ali se ispostavilo da se ne koriste (apstraktne klase, metode sa parametrima koji se ne koriste, metode sa apstraktnim nazivima).	• Dispensables [46] • Unnecessary Complexity [47]	• Smells within classes [45] • Feature-based [48]
Privremeno polje (engl. *Temporary Field*)	Polje u klasi se koristi samo privremeno u nekim slučajevima.	• Object-oriented Abusers [46] • Data [47]	• Smells between classes [45] • Structure-based [48]
Lanci poruka (engl. *Message Chains*)	Jedan objekat se oslanja na drugi objekat za dobavljanje trećeg objekta od kog dobavlja četvrti objekat i tako dalje.	• Encapsulators [46] • Responsibility [47]	• Smells between classes [45] • Dependency-based [48]
Posrednik (engl. *Middle Man*)	Klasa služi samo kao posrednik između drugih klasa.	• Encapsulators [46] • Responsibility [47]	• Smells between classes [45] • Feature-based [48]
Neprikladna intimnost (engl. *Inappropriate Intimacy*)	Dve klase imaju previše zavisnosti, što dovodi do toga da jedna klasa zna previše detalja iz druge klase.	• Couplers [46] • Inheritance, Responsibility [47]	• Smells between classes [45] • Dependency-based [48]
Alternativne klase sa različitim interfejsima (engl. *Alternative Classes with Different Interfaces*)	Klase ili metode pružaju slične funkcionalnosti, ali imaju različite interfejse.	• Object-oriented Abusers [46] • Duplication [47]	• Smells within classes [45] • Name-based [48]
Nepotpune klase u biblioteci (engl. *Incomplete Library Class*)	Klase u biblioteci ne pružaju kompletnu funkcionalnost koja je potrebna korisniku biblioteke.	• Others [46] • Library Classes [47]	• Smells between classes [45] • Feature-based [48]
Klasa podataka (engl. *Data Class*)	Klasa koja pretežno skladišti podatke, bez logike u metodama. Druge klase je koriste za dobavljanje podataka koje sadrži.	• Dispensables [46] • Data [47]	• Smells between classes [45] • Feature-based [48]
Odbačeno nasledstvo (engl. *Refused Bequest*)	Klasa naslednica ne koristi nasleđene podatke i metode.	• Object-oriented Abusers [46] • Inheritance [47]	• Smells between classes [45] • Structure-based [48]

Miris koda	Definicija	Kategorija
Komentari (engl. *Comments*)	Komentari sami po sebi ne predstavljaju problem, ali često ukazuju na kod koji je nejasan i loše napisan, zbog čega se javlja potreba za objašnjenjem kroz komentare.	• Others [46] • Measured smells [47] • Smells within classes [45] • Structure-based [48]

Mantyla i saradnici [46] su definisali kategorije na osnovu *efekata koje mirisi imaju na razvoj softvera*:

1. *Dispensables* kategorija predstavlja nepotreban kod koji treba ukloniti ili ga učiniti korisnijim dodavanjem odgovornosti.
2. *Bloaters* kategorija predstavlja velik kod kojim je teško rukovati.
3. *Change-preventers* mirisi znatno otežavaju izmenu softvera.
4. *Couplers* kategorija predstavlja visok stepen spregnutosti što se kosi sa objektno-orijentisanim principima.
5. *Object-oriented abusers* mirisi ukazuju na nepravilnu upotrebu objektno-orijentisanih principa.
6. *Encapsulators* kategorija se bavi mehanizmima komunikacije i enkapsulacije.
7. *Others* kategorija obuhvata mirise koji se ne mogu smestiti u prethodno opisane kategorije.

Sledeću kategorizaciju, na osnovu *zajedničkih koncepata* koji vežu grupisane mirise, predstavio je Wake [47]:

1. *Data* kategorija obuhvata mirise koji ukazuju na delove koda koji sadrže samo podatke, bez značajnog ponašanja (funkcionalnosti).
2. *Responsibility* kategorija predstavlja mirise koji se javljaju kada klase ili metode nemaju odgovarajuću odgovornost.
3. *Unnecessary Complexity* obuhvata mirise koji ukazuju na nepotrebnu kompleksnost u softveru.
4. *Measured smells* kategorija predstavlja mirise koje je moguće otkriti pomoću strukturnih metrika, najčešće dužine.
5. *Duplication* kategorija obuhvata mirise koji se javljaju kada se kod ponavlja.
6. *Condition Logic* obuhvata mirise koji ukazuju na zamenu objektno-orijentisanih mehanizama sa uslovnim blokovima.
7. *Inheritance* obuhvata mirise koji se javljaju kada nasleđivanje nije implementirano u skladu sa objektno-orijentisanim principima.
8. *Accomodating Change* kategorija ukazuje na probleme prilikom izmena u kodu.
9. *Library Classes* obuhvata mirise koji ukazuju na nepotpune biblioteke koje se koriste u softveru.

Lacerda i saradnici [45] su podelili mirise u dve kategorije na osnovu *lokacije*:

1. *Smells within classes* kategorija obuhvata mirise koji se nalaze unutar klasa i za čiju je analizu dovoljno posmatrati izolovani isečak koda.

2. *Smells between classes* kategorija obuhvata mirise koji se nalaze između klasa i za čiju je detekciju potrebno analizirati posmatrani isečak koda i povezane isečke kode. Tačnije, potrebno je analizirati dizajn softverskog sistema.

Poslednju kategorizaciju koja će biti obrađena su sastavili Rasool i Arshad [48] na osnovu *zajedničkih koncepata* koje vežu grupisane mirise:

1. *Mass-based* predstavlja grupu mirisa koje odlikuje izrazito velik ili mali broj objekata, metoda i parametara.
2. *Feature-based* grupiše mirise koji ukazuju na nepotpuno, neadekvatno ili nepostojeće ponašanje.
3. *Dependency-based* kategorija sadrži mirise koji ukazuju na spregnutost ili ponavljanje koda.
4. *Structure-based* predstavlja grupu mirisa koji su indikatori neprikladne organizacije koda.
5. *Name-based* grupiše mirise nastale zbog neprikladnih naziva.

Tabela 2.2 Sumarizacija kategorija definisanih za mirise koda.

Autori kategorizacije	Kategorizacija zasnovana na	Broj kategorija	Broj Fowlerovih mirisa koje kategorizuju (ukupno 22)
Mantyla i saradnici [46]	Efekti na razvoj softvera	7	22
Wake [47]	Zajednički koncepti	9	22
Lacerda i saradnici [45]	Lokacija mirisa	2	21
Rasool i Arshad [48]	Zajednički koncepti	5	22

2.1.1 Analiza štetnosti i uticaja mirisa koda

Nije svaki Fowlerov miris jednako štetan za kvalitet softvera. Pojedini mirisi koda mogu imati neznatan uticaj na održivost, dok drugi mogu značajnije otežati održavanje softvera. U nastavku će biti razmotrena štetnost i uticaj Fowlerovih mirisa kroz nekoliko aspekata:

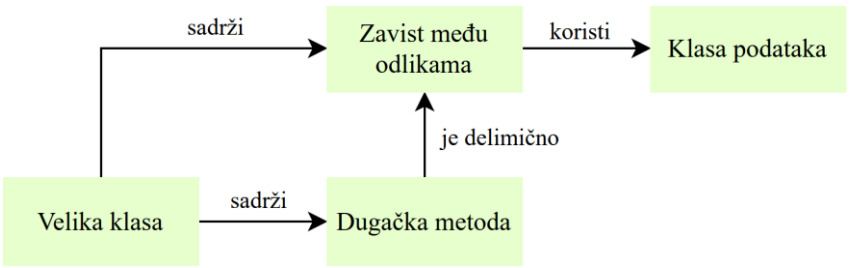
1. Rasprostranjenost mirisa u softveru
2. Sklonost softvera greškama ili izmenama zbog prisustva mirisa koda
3. Koegzistencija sa drugim mirisima
4. Složenost detekcije mirisa
5. Zastupljenost mirisa u postojećim istraživanjima

Azeem i saradnici [12] i Palomba i saradnici [24] su u svojim istraživanjima istakli da zajednica treba da prioritizuje štetne mirise koda koji se često javljaju u softveru. Među najrasprostranjenijim mirisima su *dugačka metoda* koja se javila u 84%, *velika klasa* u 65% i *odbačeno nasledstvo* u 58% analiziranih verzija softvera [24]. *Zavist među odlikama* se javila u 50% verzija softvera [24], dok *klasa podataka* nije pokrivena tim istraživanjem.

Palomba i saradnici [24] su izdvojili *veliku klasu* kao izrazito štetnu za kvalitet softvera, pošto se za klase koje pate od mirisa *velika klasa* beleži povećanje podložnosti izmenama za 283%. U slučaju *odbačenog nasledstva* je istaknuto povećanje izmena za 65%. Pored toga, softver postaje podložan greškama kada se u kodu nalaze *velika klasa*, *zavist među odlikama* i

odbačeno nasledstvo. Podložnost greškama je veoma izražena u slučaju *zavisti među odlikama* – uočeno je da je softver koji pati od ovog mirisa 8 puta više podložan greškama nego softver koji ne pati od ovog mirisa. Pored toga, istraživanja [49] i [50] ističu da prisustvo *dugačke metode* dovodi do većeg broja grešaka u softveru.

Istraživanja pokazuju da postojanje više mirisa koda u istom softveru dodatno čini softver podložnim izmenama [24] i teškim za održavanje [51]. Pretpostavlja se da koegzistencija mirisa značajno otežava programerima razumevanje koda [52]. Palomba i saradnici [31] su pokazali da je koegzistencija mirisa rasprostranjen problem i identifikovali su 6 najčešćih mirisa koji se javljaju zajedno. *Dugačka metoda*, *velika klasa, zavist među odlikama* i *odbačeno nasledstvo* se često pojavljuju sa drugim mirisima koda. U knjizi [53] je takođe obrađeno zajedničko pojavljivanje mirisa koda. Slika 2.1 prikazuje odnose između mirisa *dugačka metoda*, *velika klasa*, *klasa podataka* i *zavist među odlikama*. Na primer, metode koje pate od *zavisti među odlikama* često koriste podatke iz klasa koje pate od *klase podataka*. Dakle, *klasa podataka* obezbeđuje podatke za metodu, a metoda „zavidi klasi na podacima" jer koristi njene podatke, te pati od *zavisti među odlikama*.



Slika 2.1 Koegzistencija mirisa *zavist među odlikama*, *klasa podataka*, *velika klasa* i *dugačka metoda* [53]. *Velika klasa* često sadrži *dugačke metode*, čiji delovi mogu patiti od *zavisti među odlikama*. Konačno, metode koje pate od *zavisti među odlikama* često koriste podatke iz *klasa podataka*.

Sistematski pregled literature koji se bavi detekcijom mirisa zasnovanom na metrikama [14] pokazao je da se različiti pristupi za detekciju *dugačke metode* i *velike klase* oslanjaju na sličan skup metrika. Većina metrika se odnosi na dužinu i kompleksnost koda. Stoga, za *dugačku metodu* i *veliku klasu* je moguće definisanje kvalitetnih anotacionih smernica. Suprotno tome, za *klasu podataka*, *zavist među odlikama* i *odbačeno nasledstvo* je izraženo najmanje slaganje po pitanju metrika koje bi se koristile za detekciju [14]. Ovo ukazuje na neslaganje među anotatorima po pitanju karakteristika koda koje bi trebalo analizirati kada su u pitanju ova tri mirisa koda. Rezultati ovog istraživanja bi mogli doprineti zajednici u otkrivanju relevantnog skupa karakteristika koda za analizu *klase podataka*, *zavisti među odlikama* i *odbačenog nasledstva*.

Odabir mirisa je takođe uslovljen postojećim empirijskim istraživanjima pomoću kojih se prikupljaju viđenja programera po pitanju mirisa koda u praksi. Kako bi se mogle definisati kvalitetne smernice za anotatore, neophodno je analizirati definicije, opažanja programera i primere koji se nalaze u literaturi. Najveći broj empirijskih istraživanja se bavi mirisima *dugačka metoda* i *velika klasa* [14]. Nasuprot tome, za mirise *klasa podataka*, *zavist među odlikama* i *odbačeno nasledstvo* ne postoji velik broj istraživanja koji se bave detekcijom i zapažanjima programera. Nijedna od 10 potpuno ručnih anotacionih procedura [17] ne pokriva sva tri mirisa. Tri procedure su upotrebljene za anotaciju *klase podataka* [26][27][28], a pet procedura je anotiralo *zavist među odlikama* [27][28][29][30][31]. Međutim, samo jedno istraživanje se bavilo anotiranjem *odbačenog nasledstva* [31]. Manjak postojećih istraživanja

otežava prikupljanje i sastavljanje anotacionih smernica koje bi se koristile za sinhronizaciju anotatora. Prema tome, rezultati ovog istraživanja koji bi uključivali *klasu podataka*, *zavist među odlikama* i *odbačeno nasledstvo* bi doprineli zajednici istraživača koji se bave mirisima koda. Sa druge strane, za mirise *dugačka metoda* i *velika klasa* je moguće sastaviti kvalitetne anotacione smernice na osnovu brojnih empirijskih istraživanja.

2.1.2 Primeri odabranih mirisa

U nastavku su definisani mirisi koda koji su najinteresantniji za ovo istraživanje. Uz svaku definiciju je opis karakteristika mirisa koda i primer koda koji pati od datog mirisa.

Dugačka metoda

Metoda u kojoj je otkriveno prisustvo mirisa *dugačka metoda* ima velik broj linija koda, te bi se mogla rasparčati na kraće metode kako bi se unapredili čitljivost i održavanje koda [11][54]. *Dugačke metode* ukazuju na kompleksan kod koji je teško razumeti. Vrlo često ovakve metode implementiraju nekoliko funkcionalnosti, tj., imaju više odgovornosti [11], čime krše princip jedinstvene odgovornosti (engl. *Single Responsibility Principle* – SRP). Ovaj princip je jedan od najvažnijih principa objektno-orijentisanog programiranja koji definiše da svaka komponenta u sistemu (npr. metoda ili klasa) treba da ima jednu odgovornost. Listing 2.1 prikazuje metodu u kojoj se nalazi miris *dugačka metoda*.

```
public void showChapter(Chapter chapter, Tutorial tutorial) {
    if (chapter.orderIndex == 1) {
        this.redirectToRoot();
    }

    List <Item> higherChapters = new ArrayList<>();
    List <Item> lowerTutorials = new ArrayList<>();
    for (Item item : chapter.getItems()) {
        if (item.isHigher()) {
            higherChapters.add(item);
        } else {
            lowerTutorials.add(item);
        }
    String title = chapter.getTitle();
    String description = chapter.getDescription();

    if (chapter.getLowerItem() != null) {
        this.nextPath = this.getChapterPath(this.tutorial, chapter.getLowerItem());
        this.nextLink = this.nextPath;
    } else if (tutorial.getPathId() != null && !lowerTutorials.isEmpty()) {
        this.nextPath = this.getTutorialPath(lowerTutorials.get(0));
        this.nextLink = this.nextPath;
    } else if (chapter.isLast() && tutorial.getFootLinks().isEmpty()) {
        this.nextPath = tutorial.getFootLinks().getUrl();
    }

    if (higherChapters.isEmpty()) {
        this.prevLink = this.getChapterPath(tutorial, higherChapters);
    } else if (tutorial.getPathId() && tutorial.getHigherItem() != null) {
        Tutorial higherTutorial = tutorial.getHigherItem();
        Chapter
lastChapter=higherTutorial.getChapters().get(higherTutorial.getChapters().size()-1);

        if (lastChapter.getOrderIndex() == 1) {
            this.prevLink = this.getTutorialPath(higherTutorial);
        } else {
            this.prevLink = this.getChapterPath(higherTutorial, higherChapters);
        }
    }

    this.storeProps = new StoreProps {
        “Chapter“, chapter.getId(),
        this.signedIn ? currentUser.getCheckboxesFor(chapter) : new ArrayList<>()
    };

    this.setStore();

    this.modal = this.getModalChapter(chapter) != null ? this.getModalChapter(chapter) :
                      this.getModalTutorial(tutorial);
}
```

Listing 2.1 Primer metode showChapter zahvaćene mirisom dugačka metoda. Metoda je dugačka, sadrži kompleksne izraze i ima više odgovornosti. Različite odgovornosti su naglašene različitim bojama.

Prikazana metoda `showChapter` ima nekoliko odgovornosti:

1. Postavlja vrednost `this.nextPath`
2. Postavlja vrednost `this.prevLink`
3. Postavlja vrednost `this.storeProps`
4. Postavlja vrednost `this.modal`

Refaktorisanjem metode se nabrojane odgovornosti mogu smestiti u nove metode (`setNextPathLink`, `setPrevLink`, `setStore` i `getModal`) i pozvati iz posmatrane metode. Na taj način se povećava čitljivost koda i omogućava lakše održavanje. Refaktorisana metoda `showChapter` se može videti na Listingu 2.2.

```
public void showChapter(Chapter chapter, Tutorial tutorial) {
    if (chapter.getOrderIndex() == 1) {
        this.redirectToRoot();
    }

    this.setNextPathLink(chapter, tutorial);
    this.setPrevLink(chapter, tutorial, this.getHigherChapters(chapter));
    this.setStore(this.getStoreProps(chapter, currentUser));

    this.modal = this.getModal(chapter, tutorial);
}
```

Listing 2.2 Refaktorisana metoda showChapter ne pati od mirisa *dugačka metoda*.

Velika klasa

Miris *velika klasa* ukazuje na klase koje krše princip jedinstvene odgovornosti (SRP) pošto implementiraju više odgovornosti. *Velike klase* najčešće imaju velik broj polja i metoda, kao i velik broj linija koda [11]. Pošto klasa ima više odgovornosti, metode i polja nisu semantički povezani, te klasa ima nisku koheziju. Sa druge strane, ovakva klasa često ima mnogo veza, tj., spregnuta je sa drugim klasama u sistemu.

Primer *velike klase* se može videti u klasi `org.apache.tools.ant.Project` u projektu *Apache Ant*[8]. Klasa implementira više odgovornosti što se može zaključiti i iz komentara napisanih iznad klase. Klasa ima 105 metoda, što je najveći broj metoda u klasi u celom projektu (projekat broji 832 klase). Pored toga, broj linija koda iznosi 2476, pri čemu samo tri druge klase u projektu imaju veći broj linija koda. Za kraj, strukturna metrika za kompleksnost WMC (engl. *Weighted Method Count*) ima vrednost 108[9], što ukazuje na značajnu kompleknost koda (pravila za detekciju zasnovana na strukturnim metrikama postavljaju granicu od 20 ili 48 za detekciju velike klase [14]), iz čega se zaključuje da je ovu klasu teško održavati [55].

Velike klase se refaktorišu izdvajanjem koda u nove klase [11]. Jedan pristup podrazumeva identifikovanje odgovornosti, grupisanje polja i metoda koji rade zajedno na implementaciji odgovornosti i izdvajanje grupisanih komponenata u nove klase. Pored toga, polja ili parametri koji se često javljaju zajedno se mogu izdvojiti u zasebne klase. Takođe, moguće je analizirati druge klase i identifikovati koji skup metoda svaka od tih klasa koristi. Na osnovu te analize se određene grupe metoda mogu grupisati i izdvojiti u zasebne interfejse.

Klasa podataka

Klasa podataka predstavlja klasu koja pretežno sadrži podatke, bez značajne logike. Klasa čije metode ne sadrže relevantne funkcionalnosti za obradu podataka koji se nalaze u njoj krši principe objekto-orijentisanog dizajna, poput enkapsulacije. Takođe, druge (klijentske) klase u sistemu koriste podatke iz *klase podataka* [53].

Listing 2.3 prikazuje klasu koja je zahvaćena mirisom *klasa podataka*. Klasa `Phone` ne implementira značajnu logiku u metodama, već sadrži polja, `get` i `set` metode. `Get` metode koje se često pozivaju iz klijentskih klasa kako bi se dobavili podaci sačuvani u poljima klase ukazuju da postoji potreba za premeštanjem polja u klase koje ih i koriste. Sa druge strane, `set` metode koje se često pozivaju iz klijentskih klasa nagoveštavaju da se podacima posmatrane klase u najvećoj meri upravlja iz klijentskih klasa.

---

[8] Klasa `org.apache.tools.ant.Project` se može pronaći na sledećem linku - https://github.com/apache/ant/blob/rel/1.8.3/src/main/org/apache/tools/ant/Project.java

[9] Vrednost zavisi od implementacije računanja WMC metrike. Vrednost 108 dobijena je sabiranjem ukupnog broja uslova, petlji, logičkih operatora, ternarnih operatora i blokova za obradu izuzetaka.

```
public class Phone() {
    private String number;
    private String areaCode;
    private String prefix;

    public Phone() {}

    public String getNumber() {
        return this.number;
    }
    public String getAreaCode() {
        return this.areaCode;
    }
    public String getPrefix() {
        return this.prefix;
    }

    private void setNumber(String number) {
        this.number = number;
    }
    private void setAreaCode(String areaCode) {
        this.areaCode = areaCode;
    }
    private void setPrefix(String prefix) {
        this.prefix = prefix;
    }
}
```

Listing 2.3 Primer klase Phone zahvaćene mirisom *klasa podataka*. Klasa sadrži polja (naglašeno žutom) i samo metode za pristup poljima (zeleno) i metode za izmenu polja (plavo).

Prilikom refaktorisanja *klase podataka* je potrebno spojiti podatke i metode koje rade sa tim podacima [53]. Zatim je potrebno preusmeriti klijentske klase koje su direktno koristile podatke iz *klase podataka* da koriste metode klase koja sada spaja podatke i logiku. Ovo se može postići na nekoliko načina:

1. Utvrđivanje funkcionalnosti koje se mogu izdvojiti iz klijentskih klasa i prebacivanje u *klasu podataka*. Na ovaj način klasa dobija metode koje implementiraju logiku i upravljaju podacima koji se nalaze u istoj klasi.
2. Ako *klasa podataka* ima samo par klijentskih klasa i ne sadrži nikakvu logiku, podaci se mogu prebaciti u klijentske klase koje sa njima i rade. Nakon toga, *klasa podataka* može da se obriše.

Klasa podataka može biti klasa koja sadrži logiku, ali ima i mnogo podataka koje koriste klijentske klase. U tom slučaju je moguće refaktorisati samo deo klase. Delovi klase (podaci) se izdvajaju u novu klasu, kojoj se pridružuju metode iz klijentskih klasa koje su upravljale podacima. Zatim nova klasa pruža metode klijentima, a prvobitna *klasa podataka* više ne sadrži podatke kojima direktno upravljaju klijentske klase.

Zavist među odlikama

Metodu koja pati od mirisa *zavist među odlikama* karakteriše mnoštvo poziva metoda i atributa druge klase umesto da pretežno koristi metode i podatke klase u kojoj se nalazi. Zbog toga je izražen visok stepen spregnutosti između metode koja pati od *zavisti među odlikama* i klasa koje ta metoda koristi. Ovo dovodi do čestih promena u metodi kada se menjaju klase koje ona koristi [56].

Na Listingu 2.4 se nalazi metoda `getMobilePhoneNumber` zahvaćena mirisom *zavist među odlikama*. U primeru, metoda `getMobilePhoneNumber` pripada klasi `Customer`. Međutim,

posmatrana metoda ne koristi nijednu metodu iz klase kojoj pripada, već koristi metode klase Phone.

```
public class Phone {
    private final String unformattedNumber;
    public Phone(String unformattedNumber) {
        this.unformattedNumber = unformattedNumber;
    }

    public String getAreaCode() {
        return unformattedNumber.substring(0,3);
    }
    public String getPrefix() {
        return unformattedNumber.substring(3,6);
    }
    public String getNumber() {
        return unformattedNumber.substring(6,10);
    }
}

public class Customer {
    private Phone mobilePhone;

    public String getMobilePhoneNumber() {
        return "(" + mobilePhone.getAreaCode() + ") "
            + mobilePhone.getPrefix() + "-" + mobilePhone.getNumber();
    }
}
```

Listing 2.4 Metoda getMobilePhoneNumber zahvaćena mirisom *zavist među odlikama*. Metoda pristupa metodama klase `mobilePhone` (naglašeno žutom).

Kako bi se uklonila *zavist među odlikama*, potrebno je premestiti metodu iz originalne klase u klasu čije podatke i metode pretežno koristi. Na Listingu 2.5 se može videti refaktorisan kod – kod metode `getMobilePhoneNumber` je premešten u novu metodu `toFormattedString` u klasi `Phone`. Rezultat refaktorisanja je da metoda `getMobilePhoneNumber` ne poziva tri puta metode klase `Phone`, već samo jednom, čime je umanjena spregnutost ove dve klase.

```
public class Phone {
    private final String unformattedNumber;
    public Phone(String unformattedNumber) {
        this.unformattedNumber = unformattedNumber;
    }

    public String getAreaCode() {
        return unformattedNumber.substring(0,3);
    }
    public String getPrefix() {
        return unformattedNumber.substring(3,6);
    }
    public String getNumber() {
        return unformattedNumber.substring(6,10);
    }

    public String toFormattedString() {
        return "(" + getAreaCode() + ") " + getPrefix() + "-" + getNumber();
    }
}

public class Customer {
    private Phone mobilePhone;

    public String getMobilePhoneNumber() {
        return mobilePhone.toFormattedString();
    }
}
```

Listing 2.5 Refaktorisana metoda getMobilePhoneNumber samo jednom pristupa metodi klase `mobilePhone` (naglašeno žutom). Logika je prebačena iz `getMobilePhoneNumber` u metodu `toFormattedString`.

Odbačeno nasledstvo

Koncept nasleđivanja je osmišljen kako bi se podaci i metode roditeljske klase mogli koristiti u klasama naslednicama. Ukoliko klase naslednice odbijaju da koriste podatke i metode koje obezbeđuje roditeljska klasa, možemo zaključiti da postoji problem u odnosu tih klasa [53]. *Odbačeno nasledstvo* je miris koji zahvata klase naslednice koje ne koriste nasleđene funkcionalnosti [57]. Za klasu se može reći da pati od *odbačenog nasledstva* ukoliko redefiniše većinu nasleđenih metoda, ukazujući na pogrešnu hijerarhiju klasa [58].

Listing 2.6 prikazuje klasu `Child` koja pati od *odbačenog nasledstva.* Ova klasa redefiniše metodu `method1` i ne koristi nasleđene metode `method2` i `method3`.

```
class Parent {
    public void method1() { // does something }
    public void method2() { // does something }
    public void method3() { // does something }
}
class Child extends Parent {

    @Override
    public void method1() {
        // does something else
    }
}
```

Listing 2.6 Klasa Child zahvaćena mirisom *odbačeno nasledstvo*. Metoda `method1` je redefinisana (naglašeno zelenom), a `method2` i `method3` se ne koriste (žuto).

Odbačeno nasledstvo se može refaktorisati na nekoliko načina [53]:

1. Ako klasa naslednica semantički ne pripada hijerarhiji u kojoj se nalazi, treba je izbaciti iz posmatrane hijerarhije.
2. Ukoliko klase naslednice ne koriste neke metode koje roditeljska klasa nudi, potrebno je učiniti te metode privatnim i sakriti od klasa naslednica.
3. Roditeljska klasa može imati velik broj naslednica, pri čemu neke koriste jedan deo nasleđenih funkcionalnosti, a druge koriste drugi deo nasleđenih funkcionalnosti. U tom slučaju je potrebno napraviti novu roditeljsku klasu i u nju prebaciti jedan deo funkcionalnosti. Tu klasu treba da nasleđuju samo one klase naslednice koje koriste te funkcionalnosti.

2.2 Izazovi ručnog anotiranja mirisa koda

Cilj istraživanja je kreiranje rešenja za ručno anotiranje mirisa koda, koje je izazovno zbog dvosmislenih i nepreciznih definicija mirisa, vremenske zahtevnosti ručnog anotiranja, potrebe za specifičnom ekspertizom anotatora i semantičkom analizom koda. Četiri izazova opisana u nastavku teksta mogu negativno uticati na kvalitet skupova podataka, te ih je potrebno detaljno analizirati.

Ručno anotiranje mirisa koda je izazovno zbog nepreciznih definicija mirisa koda koje je teško primeniti u praksi. Prisustvo mirisa koda može varirati u zavisnosti od programskog jezika, radnog okvira i konvencija koje se koriste. Takođe, pokazano je da interpretacija definicije zavisi i od iskustva i intuicije anotatora [21]. Hozano i saradnici [21] su pokazali da anotatori

koriste različite strategije da identifikuju mirise koda, čak i kada im je predstavljena ista definicija. Te strategije su nazvali *heuristikama*, pokazujući da anotatori na osnovu jedne definicije mirisa koda mogu generisati od tri do 11 različitih heuristika. Na primer, za miris koda *dugačka metoda*, identifikovane su tri heuristike: analiza dužine metode, analiza dužine i upravljačkih struktura unutar metode i identifikovanje manjih delova unutar metode koji bi se mogli izdvojiti u zasebne metode. Dakle, prvi izazov u ručnom anotiranju mirisa predstavlja *utvrđivanje skupa heuristika* koje preciznije određuju prisustvo mirisa koda.

Hozano i saradnici [21] su istakli ključnu ulogu heuristika u postizanju saglasnosti anotatora. Cilj heuristika je da sinhronizuju razumevanje anotacionog problema od strane anotatora i pojednostave anotiranje [59]. Hozano i saradnici [21] i Schumacher i saradnici [59] su koristili tehniku kodiranja [60] kako bi preslikali nejasne definicije mirisa koda u diskretan skup heuristika. Anotatori bi trebalo da utvrde prisustvo mirisa koda primenjujući rezultujuće heuristike.

Primena heuristika podrazumeva razmatranje relevantnih *svojstava koda* koja informišu o ispunjenosti određene heuristike. Na primer, pri anotiranju *dugačke metode*, anotator treba da analizira sledeće heuristike: metoda je dugačka, metoda je složena i metoda ima više odgovornosti. Da bi se utvrdilo da li je metoda dugačka, anotator analizira strukturnu metriku „broj linija koda“, što je jedno od svojstava koda. Za procenu složenosti, može se koristiti analiza strukturnih metrika koje broje strukture u kodu koje doprinose složenosti ili se može osloniti na drugo svojstvo, kao što je apstraktno sintaksno stablo (AST), koje prikazuje uslovna grananja, petlje, ugnježdavanja i složenost izraza. Pored toga, anotator može analizirati i treće svojstvo - izvorni kod, kako bi otkrio misteriozna imena u kodu koja otežavaju razumevanje. Za analizu broja odgovornosti koje metoda ima, anotator može pregledati izvorni kod u potrazi za delovima koda odvojenim komentarima ili novim redovima. Takođe, anotator može izvršiti semantičku analizu izvornog koda kako bi utvrdio konkretne odgovornosti koje metoda ima.

Iz prethodnog primera možemo zaključiti da drugi izazov za ručno anotiranje mirisa koda predstavlja *odabir relevantnih svojstava koda* koje je potrebno analizirati zarad utvrđivanja ispunjenosti heuristika definisanih za određeni miris koda. Pored strukturnih metrika, koje se najčešće koriste za automatske alate za detekciju mirisa i izvornog koda koji se semantički analizira, za neke Fowlerove mirise je potrebno analizirati dodatna svojstva koda. Na primer, za miris *operacija sačmarom* je korisno analizirati istoriju promena [29], jer se ovaj miris odlikuje izmenama koda na više mesta u isto vreme. Prema tome, za kreiranje kvalitetnih skupova podataka mirisa koda, neophodno je utvrditi koja su svojstva koda relevantna za detekciju Fowlerovih mirisa koda.

Treći izazov ručnog anotiranja mirisa koda je *vremenska zahtevnost* neophodne analize koda. Prilikom anotiranja mirisa, anotatori tipično analiziraju *isečke koda* (engl. *code snippet*) u kojima bi trebalo da identifikuju postojanje i, opciono, intenzitet (engl. *severity*) mirisa koda. Na primer, ako se anotira miris *dugačka metoda*, posmatra se isečak koda koji sadrži metodu. Na osnovu analize tog isečka koda, potrebno je odrediti da li je u metodi prisutan miris *dugačka metoda*. Sa druge strane, ako se anotira miris *velika klasa*, analizira se isečak koda koji sadrži klasu u kojoj je potrebno odrediti prisustvo ovog mirisa. Isečak koda koji se analizira može sadržati stotine ili hiljade linija koda, te je potrebno dosta vremena i truda kako bi anotator razumeo isečak koda i anotirao mirise.

Vremenska zahtevnost ručnog anotiranja je još izraženija kada su u pitanju mirisi koda za koje je potrebno analizirati više od jednog isečka koda. Prethodno pomenuti mirisi, *dugačka metoda* i *velika klasa*, su kategorizovani kao „mirisi unutar klasa“, koji se mogu analizirati u izolovanom isečku koda [45]. Međutim, postoji i kategorija „mirisi između klasa“, u kojoj se nalaze mirisi za koje je potrebno analizirati dizajn i semantiku sistema – ne samo izolovanog isečka koda, već i drugih isečaka koji su sa njim povezani [45]. Ovoj kategoriji pripadaju mirisi *klasa podataka*, *zavist među odlikama* i *odbačeno nasledstvo*.

Za miris *klasa podataka* je potrebno analizirati isečak koda koji sadrži posmatranu klasu, kako bi se utvrdilo da li klasa sadrži pretežno podatke bez značajnih funkcionalnosti. Međutim, potrebno je analizirati i isečke koda klijentskih klasa koje pristupaju podacima posmatrane klase, sa ciljem utvrđivanja uloge posmatrane klase u odnosu na druge klase u sistemu. Zatim, tokom anotiranja *zavisti među odlikama*, anotatori treba da prouče da li posmatrana metoda pretežno koristi metode i atribute druge klase. Anotatori treba da identifikuju da li postoje ulančani pozivi metoda u izvornom kodu posmatrane metode, koji ukazuju na to da metoda možda pripada drugoj klasi. Pored toga, treba da prouče metode i atribute drugih klasa koje posmatrana metoda koristi kako bi utvrdili da li ona semantički više odgovara klasi čije podatke koristi nego klasi kojoj trenutno pripada. Za kraj, anotiranje *odbačenog nasledstva* zahteva od anotatora da utvrde da li klasa naslednica koristi nasleđene metode. Anotatori treba da prouče izvorni kod kako bi utvrdili da li posmatrana klasa redefiniše većinu nasleđenih metoda. Zatim, treba da analiziraju da li posmatrana klasa koristi nasleđene podatke. Za kraj, treba da istraže da li klase koje koriste posmatranu klasu koriste podatke nasleđene od roditeljske klase.

Prethodno opisan problem vremenske zahtevnosti ručnog anotiranja je opisan isključivo iz ugla analize izvornog koda isečaka. Međutim, ako uzmemo u obzir da je korisno analizirati i druga svojstva koda koja su relevantna za određene mirise koda, problem vremena potrebnog za anotiranje postaje izuzetno izražen.

Za kraj, anotiranje mirisa koda je izazovno i zato što *zahteva specifičnu ekspertizu*, kako bi anotatori mogli razumeti dizajn i semantiku koda. Pored znanja o razvoju softvera, arhitekturi softvera, principima dizajna i programskim jezicima, anotatori moraju posedovati duboko razumevanje o mirisima koda kako bi ih mogli efektivno anotirati. Duboko razumevanje mirisa koda omogućava anotatorima da prevaziđu površno posmatranje koda i identifikuju suptilne obrasce u kodu koji upućuju na prisustvo mirisa. Specifično znanje o mirisima čini anotatore sposobnijim za pravljenje preciznijih i značajnijih anotacija. Formiranje grupe anotatora koju čine eksperti u oblasti mirisa koda zahteva značajno finansijsko ulaganje zbog vrednosti njihovog specifičnog znanja i vremena koje treba da ulože u anotiranje.

2.3 Svojstva koda relevantna za detekciju mirisa

Jedan od prethodno opisanih izazova ručnog anotiranja (Potpoglavlje 2.2) se odnosi na odabir relevantnih svojstava koda koje treba analizirati prilikom detekcije mirisa. U ovom potpoglavlju biće predstavljeno prethodno istraživanje [61] gde su identifikovana svojstva koda, te je određena relevantnost svakog svojstva za detekciju 22 Fowlerova mirisa.

Većina alata za automatsku detekciju mirisa koda je zasnovana na pravilima koja porede strukturne metrike sa predefinisanim pragovima [12]. Ovakvi alati daju nekonzistentne rezultate, pošto stručnjaci nisu usaglašeni u odabiru strukturnih metrika za određene mirise koda, kao ni u određivanju pragova [12]. Pored toga, strukturne metrike nisu dovoljne za detekciju svih Fowlerovih mirisa koda, jer ne mogu obuhvatiti druge aspekte softvera koji bi

ukazali na prisustvo mirisa. Na primer, informacije o istoriji softvera doprinose dubljem razumevanju evolucije koda i pomažu u otkrivanju šablona koji ukazuju na prisustvo određenih mirisa, poput *operacije sačmarom* [29]. Zatim, apstraktna sintaksna stabla (engl. *Abstract Syntax Tree* – AST) daju uvid u strukturu i organizaciju koda, te omogućavaju detekciju mirisa koje nije lako uočiti analizom izvornog koda [62]. Za specifične mirise koda, poput *zavisti među odlikama*, je neophodno analizirati i veze među modulima softvera [63][64].

U prethodnom istraživanju [61] su identifikovana svojstva koda koja su relevantna za automatsku i ručnu detekciju Fowlerovih mirisa koda. Identifikovanje svojstava obuhvata svojstva koda navedena u literaturi kao često korišćena, bez obzira da li su korišćena samostalno ili u kombinaciji sa drugim svojstvima. Za identifikovanje svojstava je istražena literatura koja obuhvata nedavno objavljene pregledne radove vezane za detekciju mirisa koda [12][65][66][67], a od posebnog značaja je rad [68] koji je omogućio kreiranje inicijalnog kataloga svojstava.

Inicijalni katalog je sadržao šest svojstava relevantnih za detekciju mirisa: *intuicija i iskustvo programera*, *vrednosti metrika*, *apstraktno sintaksno stablo* (AST), *istorija promena*, *dinamičko ponašanje koda* i *postojanje drugih mirisa* [68]. Svojstvo *dinamičko ponašanje koda* je izbačeno iz kataloga, jer nije dovoljno detaljno obrađeno u postojećoj literaturi. Pored toga, fokus ovog istraživanja je detekcija pojedinačnih mirisa koda, te je svojstvo *postojanje drugih mirisa* izostavljeno iz kataloga. Preostala četiri svojstva iz inicijalnog kataloga su zadržana. U ovom istraživanju je dodato i peto svojstvo, koje se odnosi na *veze među komponentama sistema.* Konačan katalog svojstava koda obuhvata:

- Strukturne metrike
- Apstraktno sintaksno stablo
- Izvorni kod
- Istoriju promena
- Veze među komponentama sistema

Za katalog od pet svojstava su definisane relacije sa 22 Fowlerova mirisa [8]. Relacije prikazane u Tabeli 2.3 ukazuju na relevantnost svojstva za detekciju određenog mirisa. U tabeli su navedeni radovi na osnovu kojih je zaključeno da su svojstva relevantna za detekciju određenih mirisa. Međutim, nisu svi mirisi koda jednako zastupljeni u literaturi. Najveći broj radova se bavi najrasprostranjenijim i najštetnijim mirisima [69]. Za definisanje relacija između mirisa i svojstva koda za koje nije bilo moguće pronaći odgovore u literaturi, primenjena je *metoda stručnog mišljenja* (engl. *expert opinion*) [70]. Stručnjaci koji su učestvovali u *metodi stručnog mišljenja* imaju nekoliko godina iskustva u istraživanju mirisa koda [15][71].

Od 110 relacija između pet svojstava i 22 mirisa, 60% relacija se oslanja na postojeću literaturu. Preostalih 40% relacija je ishod *metode stručnog mišljenja*. Tačnije, *metoda stručnog mišljenja* je primenjena:

- u 22.7% slučajeva za izvorni kod (za pet od 22 mirisa)
- u 36.4% slučajeva za AST (za osam od 22 mirisa)
- u 4.5% slučajeva za strukturne metrike (za jedan od 22 mirisa)
- u 59.1% slučajeva za istoriju promena (za 13 od 22 mirisa)
- u 77.3% slučajeva za veze među komponentama sistema (za 17 od 22 mirisa)

Tabela 2.3 Svojstva koda u relaciji sa Fowlerovim mirisima koda. Relacije (vrednosti „Da“ i „Ne“) ukazuju da li je svojstvo relevantno za detekciju određenog mirisa koda.

Miris koda	**Svojstva koda**				
	Strukturne metrike	**AST**	**Izvorni kod**	**Istorija promena**	**Veze**
Dugačka metoda	Da[14][45][58][72]	Da [62][73][74]	Da [23][57][78]	Da [80]	Ne
Velika klasa	Da[14][45][58][72]	Da [62][74]	Da [23][57][78]	Da [29][56][80][81]	Da [86][87]
Opsesija primitivnim podacima	Da [14]	Ne	Da	Da	Ne
Dugačka lista parametara	Da [14][45][58]	Da [73][76]	Da [23][57]	Da	Ne
Skupine podataka	Da [45]	Da [75][76]	Da [75]	Da	Ne
Ponavljanje *switch* naredbe	Da [14]	Da [73][75][76]	Da [75][78]	Da	Ne
Privremeno polje	Da [14]	Da [76]	Da [78]	Ne	Ne
Odbačeno nasledstvo	Da [14][45][58]	Da [73][76]	Da [23][57][78]	Da [80]	Da
Alternativne klase sa različitim interfejsima	Da [14]	Da	Da	Ne	Ne
Paralelne hijerarhije nasleđivanja	Da [14]	Da	Da [78]	Da [29][56]	Da
Divergentne izmene	Da [14][45]	Ne	Da	Da [29][56][79]	Da [84]
Operacija sačmarom	Da [14][45]	Ne	Da [78]	Da [29][56][79][80]	Da [86]
Lenja klasa	Da [14][45]	Da [73][76]	Da [23][57][78]	Ne	Ne
Klasa podataka	Da [14][45]	Da [76]	Da [78]	Da [80][81]	Da [87]
Ponovljeni kod	Da [14][45]	Da [77]	Da [23][78]	Da [79]	Ne
Pretpostavljena uopštenost	Da [14][58]	Da [75]	Da [23][57][75]	Ne	Ne
Lanci poruka	Da [14]	Da [73][75]	Da [75][78]	Ne	Da
Posrednik	Da [14][58]	Da [75]	Da [57][75][78]	Ne	Da
Zavist među odlikama	Da [14][45][58][72]	Da [62][73][74]	Da [23][57][78]	Da [29][56][80]	Da
Neprikladna intimnost	Da [58]	Ne	Da [23][57][78]	Ne	Da [85]
Nepotpune klase u biblioteci	Ne	Ne	Da	Ne	Ne
Komentari	Da [14]	Ne	Da	Ne	Ne

Svojstvo iz inicijalnog kataloga [68], *vrednosti metrika*, je u ovom istraživanju preimenovano u *strukturne metrike*. Strukturne metrike omogućavaju kvantifikaciju strukturnih osobina koda. Kao što je već pomenuto, većina automatskih alata za detekciju mirisa koristi strukturne metrike izvučene iz programskog koda [29][45]. Skup pravila zasnovanih na strukturnim metrikama i predefinisanim pragovima za mnoge Fowlerove mirise se može pronaći u radu [14]. Pored toga, strukturne metrike se koriste i u ML modelima za detekciju mirisa [58][72].

Apstraktno sintaksno stablo pruža prikaz strukture koda, pri čemu čvorovi predstavljaju konstrukte u kodu. AST se često koristi kod tehnika zasnovanih na modelima za otkrivanje određenih obrazaca u kodu [14]. Apstraktna sintaksna stabla pružaju strukturne i semantičke informacije koje se mogu koristiti prilikom detekcije mirisa [62][73][74][75][76][77]. Miris koda *alternativne klase sa različitim interfejsima* je jedan od mirisa koji nisu pokriveni u literaturi po pitanju korišćenja AST za detekciju. Pošto ovaj miris ukazuje na klase različitih imena sa sličnim metodama, *metodom stručnog mišljenja* je zaključeno da bi AST doprinelo uočavanju sličnih šablona u posmatranim klasama. Za drugi miris koji je slabo zastupljen u literaturi u pogledu detekcije putem AST, *paralelne hijarhije nasleđivanja*, je *metodom stručnog mišljenja* zaključeno da bi se hijerarhije mogle proučavati kroz AST.

Svojstvo iz inicijalnog kataloga [68], *intuicija i iskustvo programera*, je u ovom istraživanju preimenovano u *izvorni kod*. Izvorni kod je izuzetno važan za ručnu detekciju mirisa koda gde anotatori proučavaju izvorni kod i oslanjaju se na svoju intuiciju i iskustvo kako bi otkrili mirise koda. Analiza izvornog koda pomaže u otkrivanju problema u dizajnu ili implementaciji softvera koji ukazuju na prisustvo mirisa. Istraživanja [23][57][75][78] pokazuju da programeri koriste izvorni kod kada se od njih zahteva da analiziraju mirise koda i njihov intenzitet. Za mirise *opsesija primitivnim podacima*, *alternativne klase sa različitim interfejsima*,

divergentne izmene, *nepotpune klase u biblioteci* i *komentari*, postojeća literatura ne nudi primere upotrebe izvornog koda prilikom detekcije mirisa. *Metodom stručnog mišljenja* je zaključeno da se svi Fowlerovi mirisi mogu otkriti kroz izvorni kod, gde anotatori analiziraju strukture u kodu koje ukazuju na potrebno refaktorisanje [8].

Istorija promena pruža uvid u evoluciju softvera. Informacije o promenama omogućavaju otkrivanje šablona koji ukazuju na prisustvo mirisa koda [29]. U nekoliko prethodnih istraživanja je istorija promena korišćena prilikom detekcije mirisa [29][56][79][80][81]. Ova istraživanja su posebno istakla korist istorije promena prilikom detekcije *divergentnih izmena* i *operacije sačmarom*. Postojeća literatura ne pokriva upotrebu istorije promena u detekciji *opsesije primitivnim podacima*, *dugačke liste parametara* i *skupine podataka*. Pošto je u [82] istaknuto da nabrojani mirisi vremenom rastu, *metodom stručnog mišljenja* je zaključeno da bi istorija promena mogla doprineti detekciji ovih mirisa. Pored toga, *ponavljanje switch naredbe* je miris kod kog je suštinski problem ponavljanje koda [8]. Pošto se miris *ponovljeni kod* može analizirati kroz istoriju promena [79], isto je zaključeno i za *ponavljanje switch naredbe.*

Veze među komponentama sistema se mogu koristiti za utvrđivanje određenih mirisa koda, jer ukazuju na spregnutost komponenti [63][64][83][84]. Veze među komponentama sistema mogu biti veze *nasleđivanja*. U slučaju nasleđivanja, korisne informacije za detekciju mirisa mogu biti koje klase su u hijerarhiji nasleđivanja i koji su direktni naslednici ili roditelji posmatrane klase. Zatim, veza može biti *poziv metode*, gde je korisno proučiti pozvanu metodu, klasu kojoj pozvana metoda pripada i klase iz kojih se poziva posmatrana metoda. Slično tome, proučavanje *podataka kojima se pristupa ili koji se menjaju* može biti korisno za detekciju mirisa. Veze među komponentama se mogu proučavati i na primerima gde *jedna klasa sadrži polje koje za tip ima drugu klasu*. Ovo važi i za *lokalne promenljive*, *parametre u metodama* ili *povratne vrednosti*. Ove veze ukazuju na visoku spregnutost sistema. Neke strukturne metrike pružaju uvid u veze među komponentama, poput CBO (engl. *Coupling Between Objects*) i DIT (engl. *Depth of Inheritance Tree*). Međutim, semantiku nije moguće detaljno analizirati putem ovih metrika.

Vizuelizacija veza među komponentama se koristila za detekciju mirisa koda u nekoliko prethodnih istraživanja [84][85][86][87]. Postojeća literatura ne obrađuje mirise *odbačeno nasledstvo*, *paralelne hijerarhije nasleđivanja*, *lanci poruka*, *posrednik* i *zavist među odlikama*, te su *metodom stručnog mišljenja* izvedeni zaključci o relevantnosti analize veza među komponentama prilikom detekcije ovih mirisa. Prilikom detekcije *odbačenog nasledstva* bi trebalo proučiti hijerarhiju nasleđivanja u kojoj se klasa nalazi [21]. Zatim, u istraživanju [14] je za detekciju *paralelnih hijerarhija nasleđivanja* upotrebljeno nekoliko metrika koje predstavljaju numeričke vrednosti određenih karakteristika hijerarhije u kojoj se posmatrana klasa nalazi. Za *lance poruka* je karakterističan niz poziva nad drugim objektima [21], te bi veze između tih objekata trebalo analizirati prilikom detekcije ovog mirisa. Klasa ili metoda zahvaćena mirisom *posrednik* ima reference ka drugim klasama čije metode poziva [14][75]. Anotatori bi trebalo da analiziraju veze među komponentama sistema kako bi utvrdili da li *posrednik* samo delegira posao drugim klasama ili ima značajniju ulogu u sistemu. Za kraj, metode koje pate od mirisa *zavist među odlikama* pristupaju podacima drugih klasa [21][64]. Prilikom detekcije ovog mirisa bi trebalo proučiti veze zavisnosti kako bi se odredilo kojoj klasi posmatrana metoda najviše pristupa i „zavidi“.

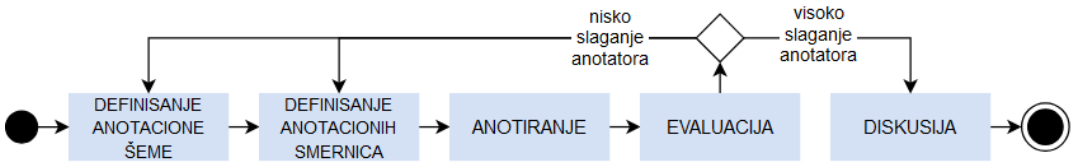
2.4 Anotacione paradigme

U Potpoglavlju 2.2 je kao jedan od izazova ručnog anotiranja predstavljeno utvrđivanje heuristika za detekciju mirisa koda, koje se javlja zbog nepreciznih i dvosmislenih definicija mirisa. U NLP oblasti se anotatori takođe susreću sa tim problemom, što može dovesti do neslaganja među anotatorima. Zbog sličnosti sa anotiranjem mirisa, u ovom potpoglavlju će biti obrađene anotacione prakse iz NLP oblasti koje se mogu primeniti pri anotiranju mirisa.

NLP zadaci često zahtevaju semantičko razumevanje teksta, što istraživači ističu kao izazovno za anotiranje zbog dvosmislenosti prirodnog jezika [39]. Mnogi NLP zadaci su subjektivni jer postoji više validnih uverenja o tome šta anotacija treba da bude. Ovakve situacije zahtevaju da se prilikom kreiranja skupa podataka razmotri uloga subjektivnosti anotatora – subjektivnost se treba ili eksplicitno ohrabriti ili eksplicitno ograničiti. U skladu sa time, pri kreiranju skupa podataka se bira jedna od dve anotacione paradigme – deskriptivna ili preskriptivna [36]. Deskriptivna paradigma ohrabruje subjektivnost anotatora u cilju obuhvatanja i modelovanja različitih mogućih razumevanja. Nasuprot tome, preskriptivna paradigma obeshrabruje subjektivnost anotatora kako bi rezultujući skup podataka sadržao isključivo jedno specifično razumevanje [36].

Preskriptivna paradigma je prikladna za izgradnju kvalitetnih ML modela [37] ili skupova podataka za poređenje različitih ML pristupa. Slika 2.2 ilustruje proces anotacije koji je usklađen sa najboljim anotacionim praksama ustanovljenim u NLP oblasti [37]. Anotacioni proces koji usvaja preskriptivnu paradigmu podrazumeva razvoj anotacione šeme i smernica koje jasno izražavaju odabrano razumevanje anotacionog zadatka, time smanjujući subjektivnost anotatora i njihova međusobna neslaganja. *Anotaciona šema* je model koji opisuje cilj anotacije i definiše informacije koje je potrebno obuhvatiti skupom podataka (na primer, moguće klasne oznake, metapodatke i relacije između klasnih oznaka i metapodataka). *Anotacione smernice* su uputstva kako se anotaciona šema primenjuje na podatke, praćena detaljnim objašnjenjima i primerima.

Na osnovu šeme i smernica, anotatori anotiraju deo skupa podataka. Nakon toga se prelazi na evaluaciju anotacija, pri čemu se izračunava slaganje anotatora (engl. *Inter-Annotator Agreement* – IAA). Ako je slaganje nisko, potrebno je revidirati šemu i/ili smernice i ponoviti anotiranje. Ako je slaganje anotatora nakon ponovljenog anotiranja visoko, prelazi se na anotaciju ostatka skupa podataka, gde se konfliktne anotacije razrešavaju diskusijom.



Slika 2.2 Proces anotacije u NLP oblasti [37].

Poput zadataka anotacije u oblasti obrade prirodnog jezika, anotiranje mirisa koda zahteva dublje razumevanje semantike programskog koda. Prethodna istraživanja beleže neslaganja među anotatorima nastala zbog dvosmislenih definicija mirisa koda i nedostatka anotacionih smernica [23]. Zbog sličnosti problema anotacije mirisa koda i prirodnog jezika, ustanovljene dobre prakse za anotaciju u NLP oblasti se mogu generalizovati na problem anotiranja mirisa

koda. Usvajanje preskriptivne paradigme i kreiranje anotacione šeme i pridruženih smernica bi doprinelo sistematičnosti procedure za ručno anotiranje mirisa koda.

2.5 Postojeće procedure za ručno anotiranje mirisa koda

U cilju sticanja uvida u postojeće procedure za ručno anotiranje Fowlerovih mirisa koda, analizirana je 21 procedura iz sistematskog pregleda literature [17]. Deset procedura propisuje da se skup podataka u potpunosti ručno anotira. Preostalih 11 procedura koriste poluautomatski pristup anotaciji skupa podataka. Iako su procedure u kojima se skup podataka ručno anotira primarne, korisno je analizirati i one delove poluautomatskih procedura koji podrazumevaju ručnu anotaciju. Utvrđeno je da sve analizirane procedure sadrže jedan ili više od sledeća četiri koraka: obuka anotatora, anotiranje mirisa koda, validacija anotacija i diskusija. Ovi koraci su opisani u Tabeli 2.4, gde su za svaki korak pobrojane procedure koje uključuju taj korak. Pored toga, istaknut je i procenat procedura koje uključuju određeni korak.

Tabela 2.4 Koraci identifikovani u anotacionim procedurama iz sistematskog pregleda literature [17].

Korak	Opis	Procedure	% procedura
1. Obuka anotatora	Inicijalna faza u anotacionoj proceduri u kojoj anotatori stiču znanje i veštine neophodne za anotaciju mirisa koda.	[13][29][30][32][33][88]	6 od 21 (28.6%)
2. Anotiranje mirisa koda a) korišćenjem automatskih alata b) ručno	Identifikovanje i beleženje prisustva i, opciono, intenziteta mirisa koda.	a) Anotacije dobijene korišćenjem automatskih alata: [13][24][88][89][90][91][92][93][94][95][96]	11 od 21 (52.4%)
		b) Ručne anotacije: [26][27][28][29][30][31][32][33][34][35]	10 od 21 (47.6%)
3. Validacija anotacija a) anotacije dobijene od automatskih alata b) ručno dobijene anotacije	Validacija anotacija dobijenih od automatskih alata ili anotatora (unakrsna provera), pri čemu se rešavaju konfliktne anotacije.	a) Validacija anotacija dobijenih od automatskih alata: [13][24][88][89][90][91][92][93][94][95][96]	11 od 11 procedura koje koriste automatske alate (100%)
		b) Validacija ručno dobijenih anotacija (unakrsna provera): [27][28][29][30][31][32][33][34][35]	9 od 10 procedura koje ručno anotiraju mirise koda (90%)
4. Diskusija	Anotatori diskutuju anotacije i razmenjuju mišljenja u cilju rešavanja nesuglasica (konfliktnih anotacija) i unapređenja anotacione procedure.	[13][24][30][31][34][88][90]	7 od 21 (33.3%)

Obuka anotatora usklađuje razumevanje mirisa koda među anotatorima, time povećavajući njihovu saglasnost i tačnost anotacija [38][39]. Iako obuka uvećava kvalitet rezultujućih skupova podataka, samo oko 28% postojećih procedura uključuje ovaj korak. Zatim, oko 52% procedura je koristilo automatske alate za anotiranje mirisa koda, većinom zasnovane na strukturnim metrikama i predefinisanim pragovima [12]. Međutim, skup korišćenih metrika i njima dodeljenih pragova nije standardizovan, što dovodi do neusaglašenih rezultata [12]. Poslednji korak, diskusija, omogućava anotatorima da analiziraju anotacije i reše nesuglasice. Pored toga, anotatori mogu poboljšati proceduru kroz diskusiju ako zaključe koja ograničenja procedure su dovela do nesuglasica. Rezultat diskusije su konzistentnije anotacije [39], čime se povećava kvalitet skupova podataka. Međutim, samo trećina postojećih procedura uključuje diskusiju.

Tabela 2.5 sažima poređenje procedura za ručnu anotaciju iz sistematskog pregleda literature [17] u pogledu ispunjenosti zahteva koji adresiraju fenomen nekonzistentnih anotacija (Tabela

1.1). Zahtev Z1 nalaže da procedura treba da uključi obuku anotatora, Z2 da anotiranje treba da vrši više anotatora kako bi postojala validacija anotacija, a Z3 da se obezbedi diskusija tj. razrešavanje konfliktnih anotacija. Međutim, mali broj postojećih ručnih anotacionih procedura uključuje obuku, validaciju i diskusiju. Od 10 procedura, samo jedna ispunjava ove zahteve [30]. Četiri anotacione procedure [26][27][28][35] ne uključuju ni obuku ni diskusiju. Pored toga, procedura [26] izostavlja i validaciju anotacija, te skupovi podataka nastali iz ove procedure mogu sadržati isečke koda koje je anotirao samo jedan anotator. Dve od ovih procedura [31][34] nisu uključile obuku anotatora, dok tri procedure [29][32][33] nisu uključile diskusiju.

Procesi obuke anotatora se značajno razlikuju u procedurama koje uključuju ovaj korak. Na primer, u proceduri [29], dva studenta master studija su analizirala definicije mirisa koda iz literature. Nije precizirano da li su studenti zajedno prošli obuku i da li su morali da usklade svoje razumevanje definicija. Sa druge strane, u procedurama [32] i [33] su studenti dobili primere mirisa koda, ali nije istaknuto da li su zajedno analizirali primere i da li su uskladili svoje razumevanje mirisa.

Razlike među procedurama se mogu videti i u koraku diskusije. U proceduri [31], dva studenta master studija su diskutovala neslaganja kako bi došli do kompromisa. Procedura [34] je imala sličan pristup koji se razlikuje u tome što tri anotatora treba da postignu kompromis diskutujući nesuglasice. Čak i ako je postignut kompromis između dva anotatora, neophodno je da se i treći složi. Ovakav pristup može značajno produžiti vreme kreiranja skupa podataka.

Iako procedura [30] obuhvata tri koraka važna za uspostavljanje kvaliteta skupova podataka, postoje izvesna ograničenja u sprovedenim koracima. Obuka anotatora zahteva od anotatora analizu definicija mirisa koda iz literature. Međutim, nije precizirano da li su anotatori analizirali i primere mirisa koda, da li su zajedno prošli kroz obuku i da li su morali da usaglase svoje razumevanje mirisa. Što se tiče validacije, jedan anotator je identifikovao isečke koda koji sadrže miris koda, dok je drugi anotator validirao postajanje mirisa koda u izdvojenim kandidatima. Na ovaj način su identifikovane lažne pozitivne anotacije, ali ne i lažne negativne anotacije, koje takođe narušavaju kvalitet skupa podataka. Diskusija koju su sproveli ima sličan nedostatak, pošto se fokusira na razrešavanje nesuglasica vezanih samo za lažno pozitivne anotacije.

Tabela 2.5 Poređenje procedura za ručnu anotaciju skupa podataka, sa fokusom na tri koraka (obuka anotatora, validacija anotacija i diskusija).

Procedura	Obuka anotatora	Validacija anotacija	Diskusija
[26]	Ne	N/A [10]	Ne
[27]	Ne	Da	Ne
[28]	Ne	Da	Ne
[29]	Definicije	Da (2 anotatora)	Ne
[30]	Definicije	Da (2 anotatora)	Razrešavanje lažnih pozitivnih anotacija
[31]	Ne	Da (2 anotatora)	Razrešavanje nesuglasica dok se svi anotatori ne slože
[32]	Primeri	Da	Ne
[33]	Primeri	Da	Ne
[34]	Ne	Da (3 anotatora)	Razrešavanje nesuglasica dok se svi anotatori ne slože
[35]	Ne	Da (4 anotatora)	Ne

[10] Autori iz [26] navode da je više anotatora učestvovalo u anotiranju, ali nije jasno istaknuto da li je više anotatora anotiralo isti isečak koda. Iz toga se ne može zaključiti da li su imali unakrsnu proveru.

Nakon poređenja procedura u pogledu definisanih zahteva, mogu se analizirati i drugi faktori koji čine rezultujući skup podataka kvalitetnijim. U nastavku će se razmotriti upotreba heuristika tokom anotiranja, nivo detalja koje skupovi sadrže i mogućnost prioritizacije mirisa na osnovu podataka u skupu.

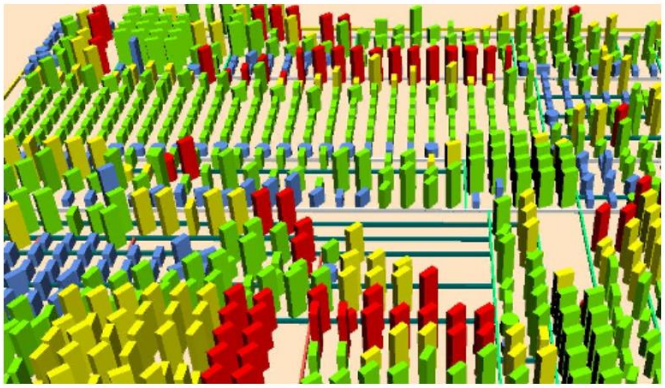
Iako su Hozano i saradnici [21] istakli ključnu ulogu heuristika u postizanju saglasnosti anotatora, nijedna od analiziranih procedura iz Tabele 2.5 nije navela da su anotatori upotrebljavali heuristike. Zatim, razmatrajući nivo detalja koji skupovi podataka sadrže, primećeno je da samo jedna postojeća ručna anotaciona procedura [27] rezultuje skupom koji sadrži sve pojedinačne labele koje su anotatori dodeljivali. Za kraj, među postojećim ručnim anotacionim procedurama, samo dve [27][34] su imale definisanu skalu intenziteta (četiri intenziteta u [27] i pet intenziteta u [34]), dok su ostale procedure koristile binarnu skalu kojom se samo označava da li je miris prisutan ili ne. Rad [17] ističe da je intenzitet mirisa koda neophodan faktor za određivanje prioriteta uklanjanja mirisa koda, što je pogodno za tačnu procenu troškova održavanja softvera.

Na osnovu sprovedene analize je zaključeno da postoji potreba za sistematizacijom procedura za ručno anotiranje mirisa koda, pri čemu je cilj kreiranje kvalitetnih skupova podataka za obučavanje ML modela za detekciju mirisa koda. *Sistematičan pristup* bi podrazumevao da procedura obuhvata korake koji obezbeđuju konzistentnost anotacija – obuku anotatora, validaciju anotacija i diskusiju.

2.6 Postojeći alati za ručno anotiranje mirisa koda

Pored anotacionih procedura, kvalitet skupa podataka može zavisiti i od alata koje anotatori koriste za anotiranje mirisa. Alati mogu uticati na kvalitet i dubinu analize softvera, preciznost određivanja intenziteta mirisa, brzinu anotiranja i lakoću razrešavanja konflikata među anotatorima. U nastavku će biti opisani postojeći alati za analizu softvera u cilju detekcije mirisa.

VERSO [83] pruža trodimenzionalni prikaz objektno-orijentisanih programa, te omogućava analizu strukturnih metrika, veza između klasa, organizacije koda po paketima i izvornog koda. Na Slici 2.3 je prikaz klasa *Xerces*[11] softvera, iscrtan na osnovu CBO strukturne metrike.

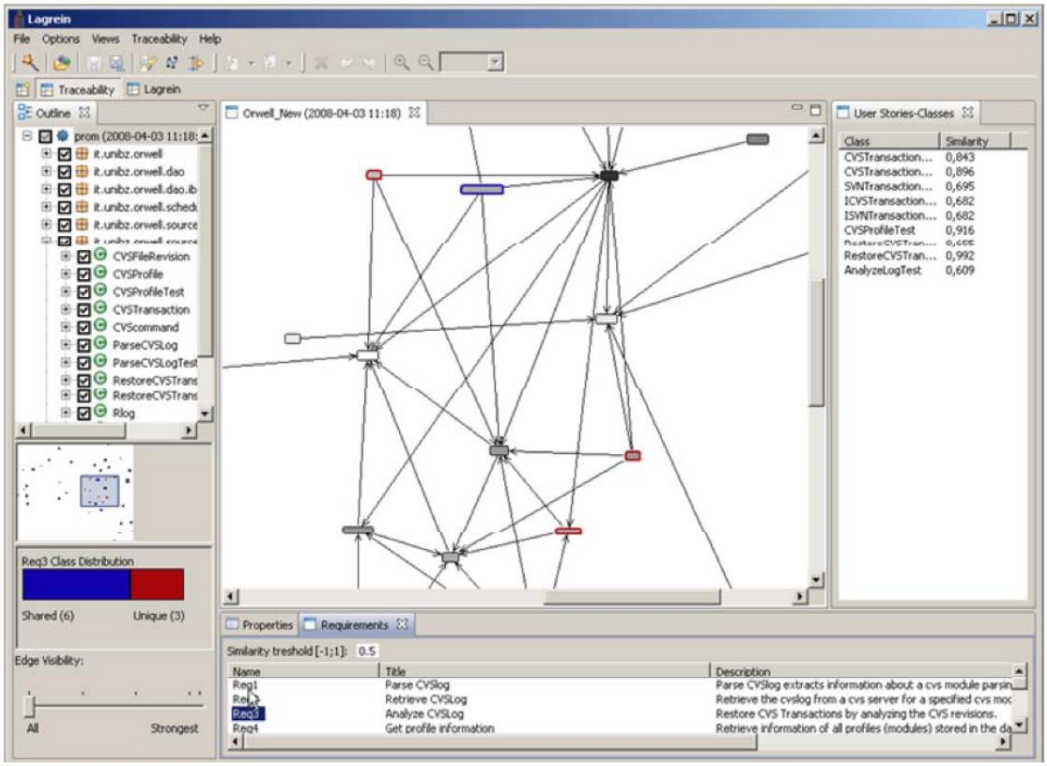


Slika 2.3 Prikaz softvera pomoću *VERSO* alata [83] na osnovu CBO strukturne metrike.

Program *Lagrein* [85] koristi strukturne metrike za veličinu, kompleksnost i spregnutost koda za vizuelizaciju softvera. Slika 2.4 predstavlja snimak ekrana iz programa *Lagrein*. Čvorovi

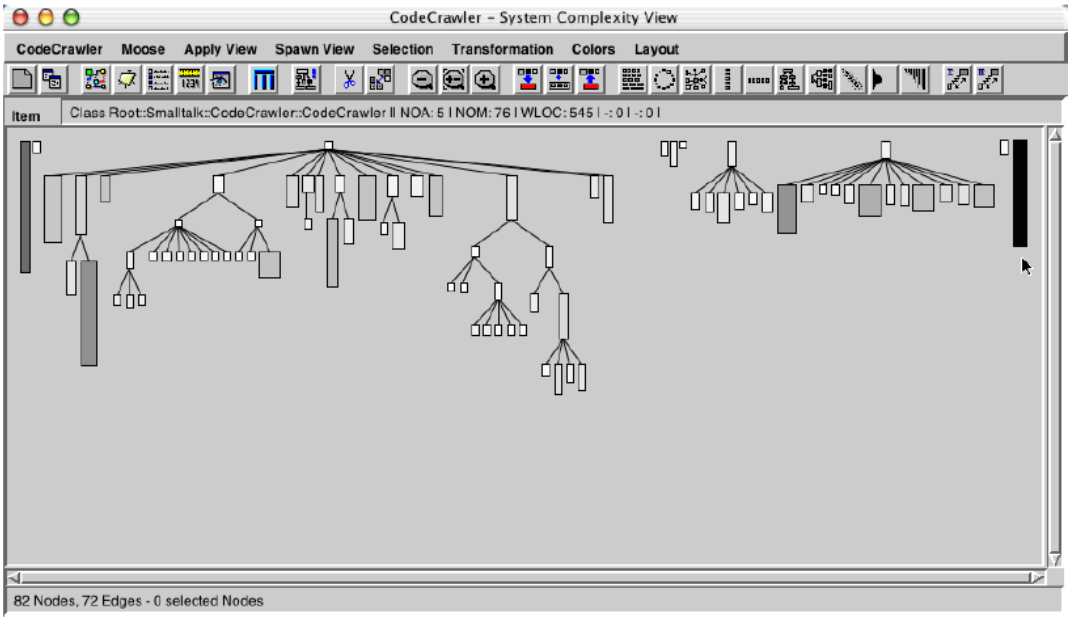
[11] http://xerces.apache.org/

grafa su klase, paketi i metode, a grane predstavljaju veze među njima, poput poziva metoda, pristupa atributima i nasleđivanja.

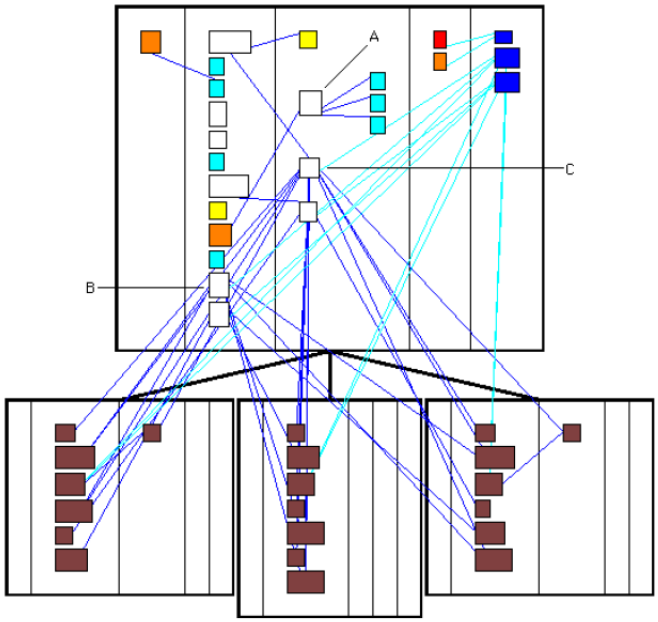


Slika 2.4 Prikaz softvera u vidu grafa pomoću *Lagrein* alata [85]. Paketi, klase i metode su predstavljene čvorovima. Grane predstavljaju veze među njima.

CodeCrawler [97] pruža vizuelizaciju softvera na nekoliko načina – sa različitim nivoima detalja, za analizu koda unutar jedne klase ili između klasa ili za analizu evolucije softvera. Na Slici 2.5 se može videti jedan od načina za prikaz klasa, pri čemu se broj atributa koristi za širinu, broj metoda za dužinu, a broj linija koda za boju pravougaonika koji predstavljaju klase. Metrike koje se koriste za prikaz pravogaonika se mogu podešavati. Na Slici 2.6 se nalazi detaljniji prikaz četiri klase koji omogućava analizu interne strukture i hijerarhije klasa.

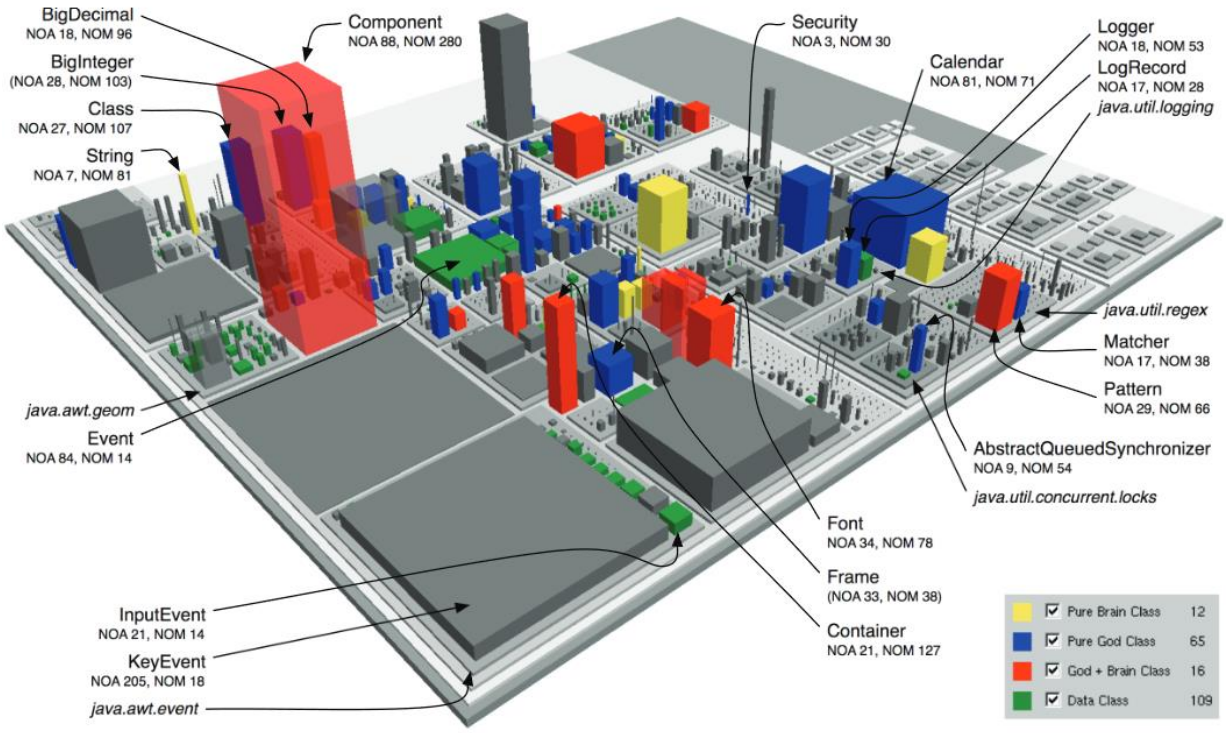


Slika 2.5 Opšti prikaz softvera pomoću *CodeCrawler* alata [97]. Dužina, širina i boja pravouoganika koji predstavljaju klase zavisi od odabranih metrika.



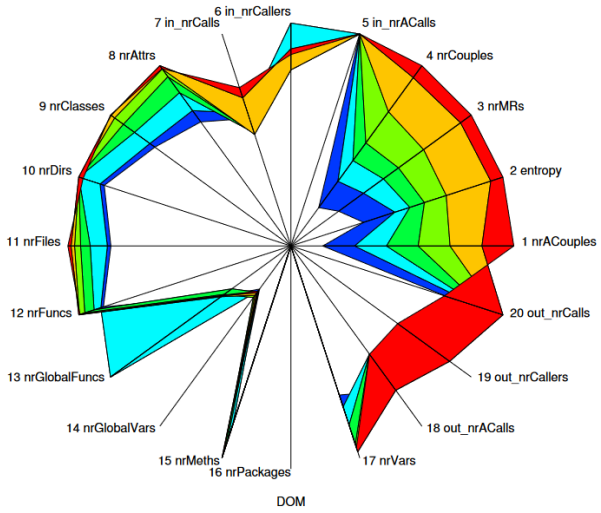
Slika 2.6 Detaljniji prikaz softvera pomoću *CodeCrawler* alata [97] prikazuje internu strukturu I hijerarhiju klasa.

U radu [87] je predstavljena vizuelizacija softvera u vidu trodimenzionalnih „gradova“. Na Slici 2.7 se mogu videti klase i interfejsi predstavljeni u vidu zgrada. Visinu zgrade određuje broj metoda (engl. *Number Of Methods* - NOM), a širinu i dužinu broj atributa (engl. *Number Of Attributes* - NOA). Pored toga, alat boji zgrade na osnovu mirisa koda otkrivenih automatski pomoću strukturnih metrika.

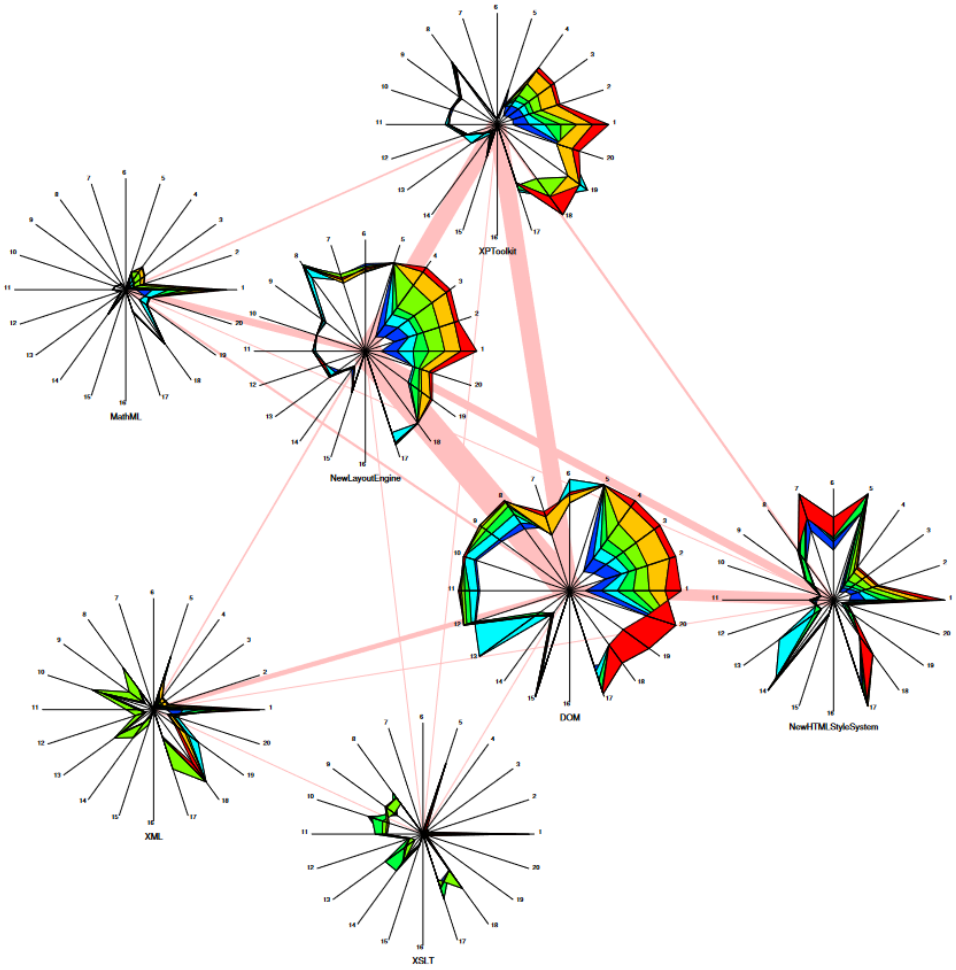


Slika 2.7 Prikaz softvera [87]. Visina, širina i dužina „zgrada“ koje predstavljaju klase zavisi od vrednosti strukturnih metrika.

Autori rada [98] su predstavili vizuelizaciju različitih strukturnih metrika tokom evolucije softvera. Na Slici 2.8 se može videti dijagram za jednu klasu sa 20 strukturnih i evolutivnih metrika u 7 verzija softvera, dok se na Slici 2.9 može videti graf od 7 takvih dijagrama (svaki dijagram predstavlja jednu klasu). Grane grafa predstavljaju spregnutost klasa.



Slika 2.8 Dijagram vrednosti 20 metrika u 7 verzija softvera [98]. Dijagram se odnosi na jednu klasu u softveru.

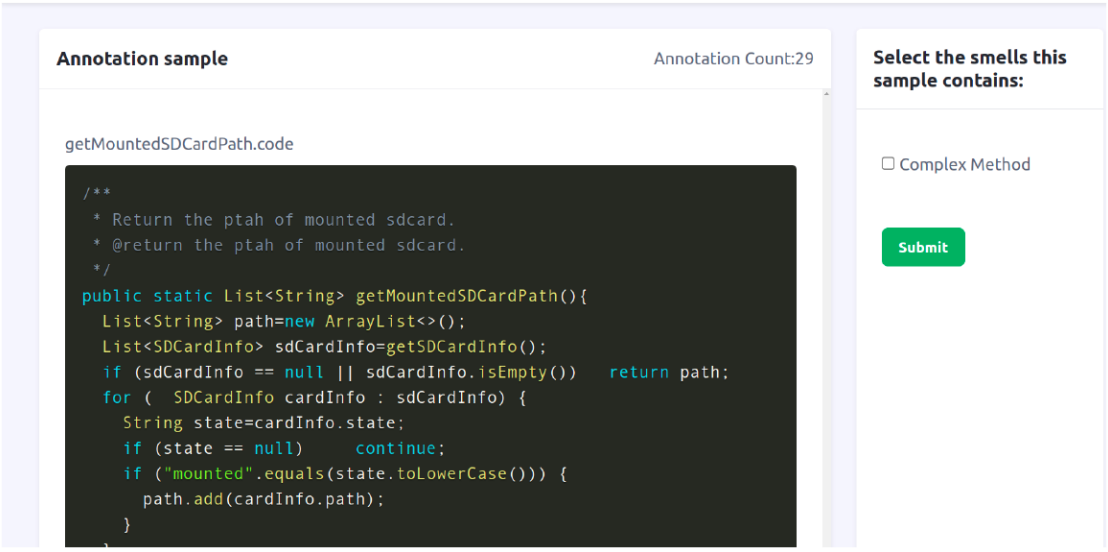


Slika 2.9 Graf za 7 klasa u 7 verzija softvera [98]. Čvor grafa predstavlja dijagram za jednu klasu na kom su prikazane vrednosti 20 metrika. Grane grafa predstavljaju spregnutost klasa.

Prethodno opisani alati omogućavaju analizu softvera, a u nekim slučajevima i automatsku detekciju, ali ih anotatori ne mogu koristiti za ručno anotiranje mirisa koda. Alat *TAGMAN*[12] omogućava analizu isečaka koda napisanih u Java programskom jeziku i beleženje anotacija. Ovaj alat je korišćen za anotiranje nekoliko mirisa koda u radu [99]. Na Slici 2.10 se može videti korisnički interfejs *TAGMAN* alata.

Međutim, alat ne pruža mogućnost definisanja anotacione šeme, te nije dovoljno fleksibilan i prilagodljiv. Nemogućnost definisanja anotacione šeme može ograničiti anotatore u anotiranju različitih vrsta mirisa koda. Sa druge strane, alat ne pruža podršku u razrešavanju konfliktnih anotacija, što može značajno uticati na kvalitet rezultujućeg skupa podataka.

[12] https://github.com/SMART-Dal/Tagman



Slika 2.10 Korisnički interfejs *TAGMAN* alata omogućava analizu izvornog koda isečka koda i beleženje prisustva mirisa koda.

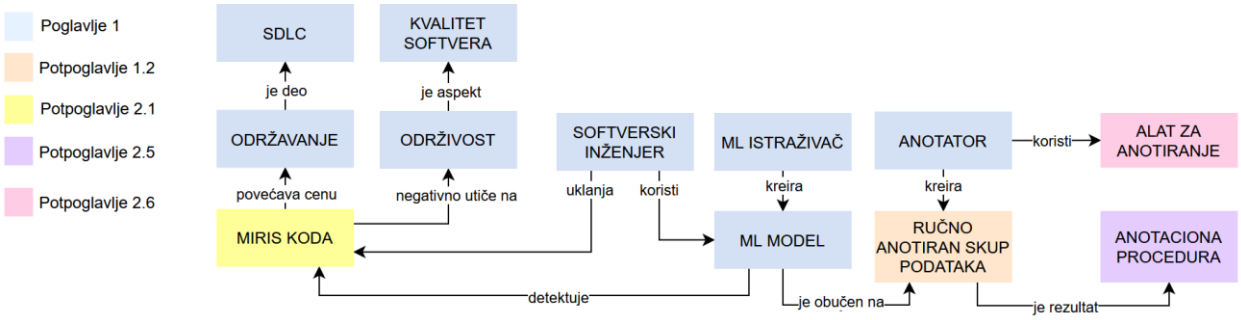
Anotiranje teksta, kao i anotiranje mirisa koda, zahteva analizu semantike i često je izazovno zbog dvosmislenih definicija [22][39]. Zbog sličnosti anotiranja mirisa koda i teksta, korisno je proučiti postojeće alate koji se koriste za anotiranje dokumenata. Pregledni rad [100] koji obrađuje alate za ručnu anotaciju dokumenata ističe prednost strukturiranih anotacija koje je lakše koristiti u mašinskom učenju, statističkoj obradi i analizi slaganja anotatora. Pored toga, anotiranje zasnovano na anotacionoj šemi obezbeđuje veće slaganje među anotatorima. Za kraj, autori rada [100] su istakli da korisnost i kompletnost alata za anotiranje može direktno da utiče na anotacioni proces, ubrzavajući ga ili usporavajući.

U literaturi se mogu pronaći razni alati za analizu softvera koji anotatorima pružaju vizuelne reprezentacije različitih informacija, poput strukturnih, semantičkih, relacionih i istorijskih. Međutim, ističe se nedostatak alata koji bi pored analize softvera omogućili i anotiranje mirisa, te pružili anotatorima podršku tokom anotacione procedure. Podrška tokom anotacione procedure bi podrazumevala mogućnost definisanja anotacione šeme i smernica, obučavanja anotatora, validiranja anotacija i analize i razrešavanja konfliktnih anotacija.

2.7 Konceptualni radni okvir problema

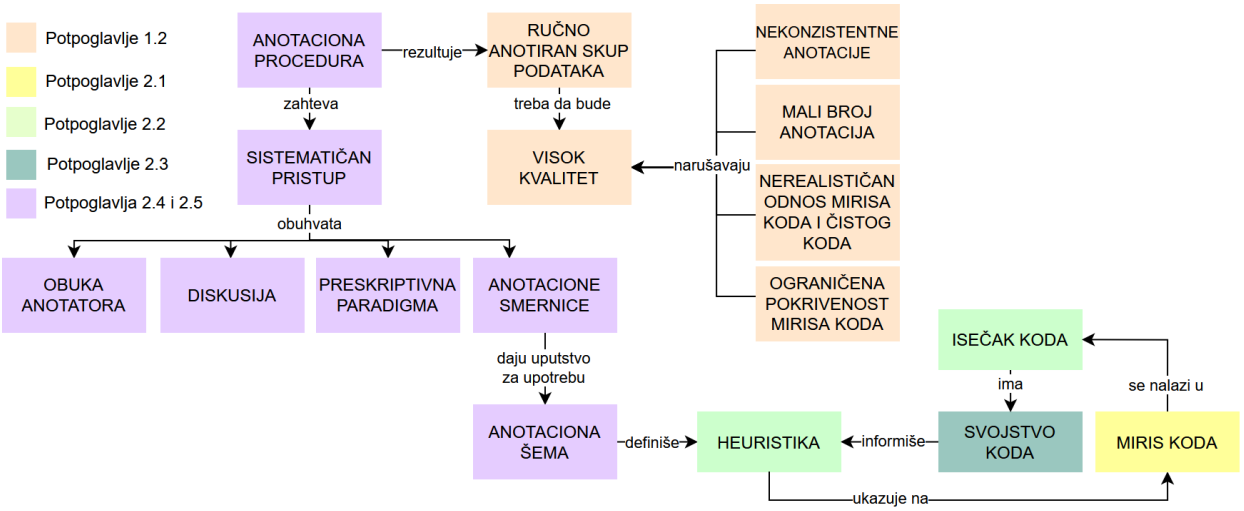
Ovo potpoglavlje rezimira analizu mirisa koda, izazova pri ručnoj detekciji, postojećih procedura za ručno anotiranje i alata za anotiranje u konceptualni radni okvir problema [70]. Konceptualni radni okvir problema, prikazan na Slici 2.11 i 2.12, definiše koncepte i veze među njima koji su ključni za razumevanje problema i projektovanje rešenja.

Slika 2.11 prikazuje koncepte koji se odnose na uticaj mirisa koda. *Miris koda* negativno utiče na *održivost* softvera, što je jedan aspekt *kvaliteta softvera*. Zbog toga mirisi koda povećavaju troškove *održavanja* softvera, što predstavlja jednu fazu *životnog ciklusa razvoja softvera* (SDLC) [9]. *Softverski inženjeri* mogu da koriste *ML modele* za uklanjanje štetnih mirisa koda [12]. *ML istraživači* koji grade ML modele mogu da koriste *ručno anotirane skupove podataka* za obučavanje ML modela [16][17]. Da bi kreirali skupove podataka, *anotatori* prolaze kroz *anotacione procedure* i koriste *alate za anotiranje*. Poglavlja koja detaljnije izlažu ove koncepte su istaknuta levo na slici.



Slika 2.11 Koncepti i veze među njima, vezani za uticaj mirisa koda. Različitim bojama su obeležena poglavlja koja detaljnije analiziraju prikazane koncepte.

Slika 2.12 sadrži koncepte koji se odnose na procedure za ručno anotiranje. *Ručno anotirani skupovi podataka* namenjeni obučavanju ML modela treba da budu *visokog kvaliteta*. Kvalitet skupova podataka narušavaju *nekonzistentne anotacije, mali broj anotacija, ograničena pokrivenost mirisa koda* i *nerealističan odnos mirisa koda i čistog koda* [25]. Izgradnja kvalitetnih skupova podataka zahteva *sistematičan pristup*, koji obuhvata *obuku anotatora* i *diskusiju* između anotatora [39]. Sistematične anotacione procedure namenjene razvoju skupa podataka za treniranje ML modela bi trebalo da prate *preskriptivnu anotacionu paradigmu* kako bi se ublažila neslaganja anotatora [36]. Konačno, sistematičan pristup podrazumeva postojanje *anotacione šeme* i pratećih *smernica* [23], koje anotatore upućuju na *heuristike* i *svojstva koda* koje treba da analiziraju prilikom anotiranja mirisa koda. Poglavlja koja detaljnije izlažu ove koncepte su istaknuta levo na slici.



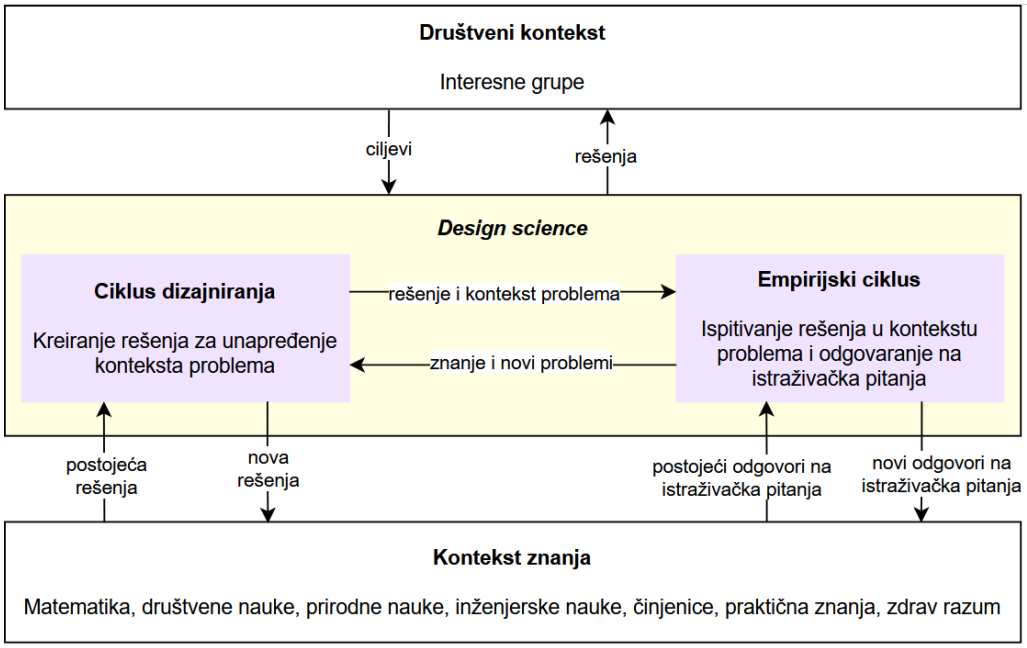
Slika 2.12 Koncepti i veze među njima, vezani za procedure za ručno anotiranje skupova podataka. Različitim bojama su obeležena poglavlja koja detaljnije analiziraju prikazane koncepte.

3. METODOLOGIJA

Istraživanje se zasniva na *design science* metodologiji koja se koristi u oblasti informacionih sistema i softverskog inženjerstva [70]. Radni okvir *design science* metodologije je predstavljen na Slici 3.1 i zasnovan je na dva ciklusa koja se iterativno ponavljaju:

1. *Ciklus dizajniranja*: Na osnovu ciljeva interesnih grupa se kreira rešenje koje treba da unapredi kontekst problema i doprinese ispunjenju ciljeva interesnih grupa.
2. *Empirijski ciklus*: Kreirano rešenje se ispituje u kontekstu problema, kako bi se odgovorilo na istraživačka pitanja. Tokom ispitivanja se stiču nova znanja o rešenju i potencijalno javljaju novi problemi, nakon čega je potrebno ponovo kreirati ili doraditi rešenje.

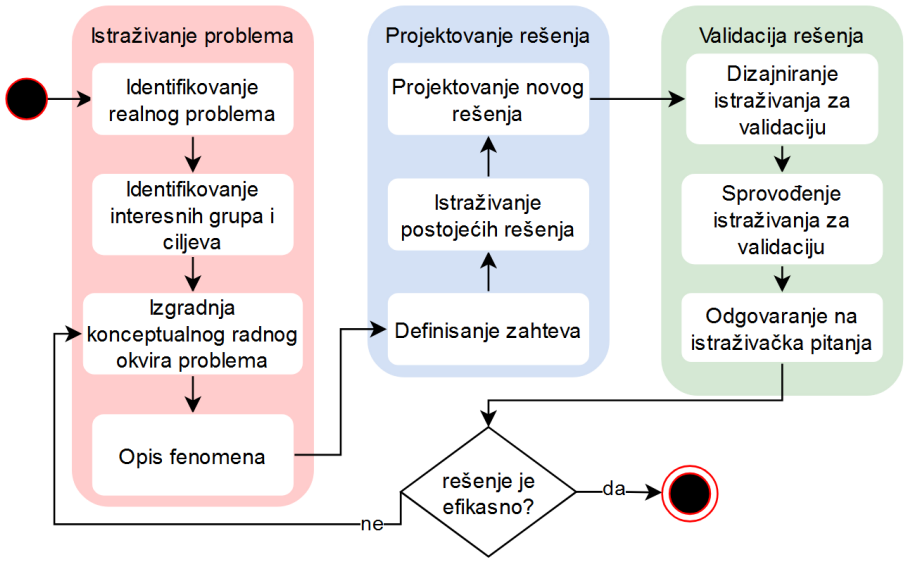
Rešenje se kreira na osnovu postojećeg znanja, dok se za ispitivanje rešenja uzimaju u obzir odgovori na postojeća istraživačka pitanja. Novo rešenje i novi odgovori na istraživačka pitanja dopunjuju kontekst znanja.



Slika 3.1 Radni okvir design science metodologije [70] koja definiše iterativno rešavanje problema na osnovu postojećeg znanja u cilju ispunjenja ciljeva interesnih grupa.

Ciklus dizajniranja se sastoji od tri faze: istraživanje problema, projektovanje rešenja i validacija rešenja. Cilj istraživanja problema je identifikovati, opisati, objasniti i evaluirati problem realnog sveta. U sledećoj fazi se projektuje rešenje koje treba da unapredi kontekst problema, dok se u fazi validacije ispituje rešenje u prototipu konteksta rešenja. Nakon validacije rešenja se ciklus ponavlja ukoliko rešenje nije efikasno.

Slika 3.2 prikazuje aktivnosti grupisane u tri faze. Potpoglavlja 3.1, 3.2 i 3.3 detaljnije opisuje aktivnosti navedenih faza. Rezultat prve iteracije ciklusa je rad [101] gde je predstavljena inicijalna anotaciona procedura koja je validirana na “mirisima unutar klasa” koji se mogu analizirati u izolovanom isečku koda [45], dok je rezultat druge iteracije rad [102] gde je predstavljena proširena anotaciona procedura koja je validirana na “mirisima između klasa” za koje je neophodno analizirati dizajn i semantiku sistema, a ne samo izolovani isečak koda [45].



Slika 3.2 Pregled aktivnosti koje čine *design science* metodologiju [70], grupisane u tri faze (istraživanje problema, projektovanje rešenja i validacija rešenja).

3.1 Istraživanje problema

Za potrebe istraživanja problema realnog sveta, analizirana je relevantna literatura i sprovedena su dodatna istraživanja koja su opisana u nastavku poglavlja. Istraženi problem je sažet i predstavljen u vidu konceptualnog radnog okvira problema u Potpoglavlju 2.7.

Identifikovanje realnog problema: Za projektovanje efikasnog rešenja za problem realnog sveta ključno je identifikovati i opisati problem, njegove uzroke i uticaj. Iz dvogodišnjeg projekta [103] čiji je cilj bio razvoj ML modela za detekciju mirisa koda je proisteklo više radova [15][104][105] koji su ukazivali na poteškoće u obuci ML modela na postojećim, javno dostupnim skupovima podataka. Analizom uzroka ovih poteškoća identifikovan je problem nekonzistentnosti anotacija u dostupnim skupovima podataka. Literatura je takođe ukazala na nedostatak široko prihvaćenog skupa podataka koji bi se koristio za dosledno merenje napretka u razvoju ML modela za detekciju mirisa koda.

Po identifikaciji problema kvaliteta skupova podataka, istraženi su izazovi ručnog anotiranja mirisa koda. Rezultati ovog istraživanja su prikazani u Potpoglavlju 2.2, gde su opisana četiri izazova: *utvrđivanje heuristika*, *odabir relevantnih svojstava koda* za detekciju mirisa, *vremenska zahtevnost* ručnog anotiranja i *potreba za specifičnom ekspertizom anotatora*. Za izazov *odabira relevantnih svojstava koda* je sprovedeno istraživanje [61], koje je predstavljeno u Potpoglavlju 2.3. Identifikovano je pet svojstava koda, te je određena relevantnost svakog od njih za detekciju Fowlerovih mirisa.

Identifikovanje interesnih grupa i ciljeva: Cilj projektovanog rešenja je da podrži interesne grupe u postizanju njihovih ciljeva. Po identifikaciji problema kvaliteta postojećih skupova podataka mirisa koda, ispitana je literatura radi identifikacije interesnih grupa na koje problem utiče. Istraženi su ciljevi identifikovanih interesnih grupa i mehanizama kako dati problem utiče na te ciljeve. Rezultat ovog istraživanja prikazan je u Potpoglavlju 1.1, gde su opisani ciljevi identifikovanih interesnih grupa: *anotatora*, *ML istraživača* i *softverskih inženjera*.

Izgradnja konceptualnog radnog okvira problema: Konceptualni radni okvir problema definiše koncepte koji su ključni za razumevanje problema i projektovanje rešenja i njihove međusobne veze. Polazni konceptualni radni okvir problema je kreiran analizom literature o uticaju mirisa koda na kvalitet koda, različitim pristupima detekcije mirisa, skupovima podataka i procedurama za njihovo anotiranje. Nakon toga, okvir je iterativno rafiniran kroz faze projektovanja rešenja i validacije rešenja. Po potrebi, konceptualni radni okvir problema se prerađuje spram novih otkrića do kojih se dolazi tokom validacije rešenja kako bi se rešili identifikovani nedostaci. Konceptualni radni okvir problema je prikazan u Potpoglavlju 2.7.

Opis fenomena: Razumevanje fenomena koji se odnose na istraživani problem je korisno za osmišljavanje rešenja koje će pomoći interesnim grupama u postizanju njihovih ciljeva. U Potpoglavlju 1.2 su opisani fenomeni koji doprinose lošem kvalitetu skupova podataka mirisa koda: *nekonzistentne anotacije*, *mali broj anotacija*, *nerealističan odnos mirisa koda i čistog koda* i *ograničena pokrivenost mirisa koda*. Objašnjeni su mehanizmi pomoću kojih se ovi fenomeni javljaju i opisan je njihov negativan uticaj na ciljeve interesnih grupa.

3.2 Projektovanje rešenja

Na osnovu uvida stečenih u fazi istraživanja problema, precizirani su zahtevi za projektovanje rešenja. U fazi projektovanja rešenja se razvija rešenje koje ispunjava definisane zahteve, čime se rešava identifikovan problem i tako doprinosi ispunjenju ciljeva interesnih grupa.

Definisanje zahteva: Definisani zahtevi moraju biti motivisani ciljevima interesnih grupa, što se opisuje kroz argumente doprinosa. Argumenti doprinosa objašnjavaju kako rešenje koje ispunjava zahteve doprinosi ciljevima. Na osnovu fenomena koji negativno utiču na ciljeve interesnih grupa (Potpoglavlje 1.2) definisani su zahtevi za planirano rešenje. Tabela 1.1 u Potpoglavlju 1.3 prikazuje listu zahteva potkrepljenu argumentima doprinosa.

Istraživanje postojećih rešenja: Analiza postojećih rešenja omogućava da novo rešenje bude utemeljeno na postojećim. Na taj način se poboljšava trenutno stanje u oblasti kroz adresiranje ograničenja identifikovanih u postojećim rešenjima. U sklopu ove aktivnosti istražene su anotacione prakse čija je namena kreiranje skupova podataka visokog kvaliteta pogodnih za obučavanje ML modela u NLP oblasti. Imajući u vidu sličnosti u anotiranju mirisa koda i prirodnog jezika, prakse koje su se pokazale korisne za NLP oblast se mogu primeniti pri anotiranju mirisa koda. Rezultat ove istrage je opisan u Potpoglavlju 2.4, gde je zaključeno da bi usvajanje preskriptivne paradigme doprinelo sistematičnosti anotacione procedure.

Zatim su istražene postojeće procedure za anotiranje mirisa koda opisane u sistematskom pregledu literature [17], a rezultati analize izloženi su u Potpoglavlju 2.5. Identifikovano je ograničenje postojećih procedura, tj., odsustvo sistematičnog pristupa koji obuhvata korake koji obezbeđuju konzistentne anotacije: obuku anotatora, validaciju anotacija i diskusiju.

Za kraj, analizirani su postojeći alati za ručno anotiranje mirisa koda. Opisi i primeri korisničkih interfejsa analiziranih alata se nalaze u Potpoglavlju 2.6, gde su istaknuta i glavna ograničenja postojećih alata. Pored analize softvera, alati uglavnom ne pružaju mogućnost anotiranja mirisa, definisanja anotacione šeme i smernica, obučavanja anotatora, validiranja anotacija i analize i razrešavanja konfliktnih anotacija.

Projektovanje novog rešenja: Ukoliko postojeća rešenja ne ispunjavaju definisane zahteve, potrebno je projektovati novo rešenje. Cilj ovog istraživanja je projektovanje rešenja za ručno anotiranje Fowlerovih mirisa koda [8] koji ispunjava zahteve definisane u Tabeli 1.1 (potglavlje 1.3). Rešenje je predstavljeno u Poglavlju 4 i sastoji se od preskriptivne anotacione procedure i alata za anotiranje.

3.3 Validacija rešenja

Potrebno je osmisliti postupak validacije rešenja putem kog je moguće utvrditi da li rešenje doprinosi ciljevima interesnih grupa. Na osnovu rezultata validacije je potrebno proceniti efikasnost rešenja i identifikovati prostor za dalje unapređenje.

Projektovanje i sprovođenje istraživanja za validaciju: Eksperiment u kom se prototip rešenja testira u simulaciji konteksta (engl. *single-case mechanism experiment*) je koristan za validaciju rešenja, jer omogućava analizu efekata dobijenih u kontrolisanom eksperimentu. Planiran je dizajn eksperimenta koji se sastoji od dve faze. U prvoj fazi će grupa od tri anotatora primeniti projektovano rešenje da anotira sledeće „mirise unutar klasa“: *dugačka metoda* i *velika klasa*. U drugoj fazi će ista grupa anotatora anotirati sledeće „mirise između klasa“: *klasa podataka*, *zavist među odlikama* i *odbačeno nasledstvo*.

Za ovo istraživanje je od kategorizacija opisanih u Potpoglavlju 2.1 odabrana kategorizacija mirisa po lokaciji [45], pošto ručno anotiranje mirisa podrazumeva analizu isečaka koda. Odabrana kategorizacija [45] definiše da se „mirisi unutar klasa“ mogu analizirati u izolovanom isečku koda, dok se za „mirise između klasa“ moraju analizirati dizajn i semantika sistema (ne samo izolovanog isečka koda, već i drugih isečaka koji su sa njim povezani). Pošto je analiza jednog isečka jednostavnija i vremenski manje zahtevna, za prvu fazu validacije su odabrani „mirisi unutar klasa“. Nakon prve faze validacije i dorade rešenja, moguće je validirati rešenje na mirisima koji su složeniji i vremenski zahtevniji za analizu – „mirisima između klasa“.

Pored različitih kategorija kojima mirisi pripadaju, sledeći faktori su takođe uticali na odabir mirisa:

- Rasprostranjenost mirisa u softveru
- Sklonost softvera greškama ili izmenama zbog prisustva mirisa koda
- Koegzistencija sa drugim mirisima
- Složenost detekcije mirisa
- Zastupljenost mirisa u postojećim istraživanjima

U Tabeli 3.1 su sumirani razlozi za odabir ovih pet mirisa, a u Potpoglavlju 2.1.1 se nalazi opširnija analiza.

Odabir isečaka koda za anotaciju je predstavljen u Potpoglavlju 5.1, a anotaciona šema i smernice koje su anotatori koristili u Potpoglavlju 5.2. Rezultujući skup podataka je opisan u Potpoglavlju 5.3.

Tabela 3.1 Sumirani razlozi za odabir mirisa za validaciju rešenja.

Odabrani miris koda	Obrazloženje odabira	Faza validacije
Dugačka metoda	• veoma rasprostranjen (u 84% analiziranih primera) [24] • čini softver sklonim greškama [49][50] • često se javlja u kombinaciji sa drugim mirisima [31][106] • velik broj empirijskih istraživanja omogućava definisanje kvalitetnih anotacionih smernica [14]	1
Velika klasa	• veoma rasprostranjen (u 65% analiziranih primera) [24] • čini softver sklonim izmenama i greškama [24][45][107] • često se javlja u kombinaciji sa drugim mirisima [31] • velik broj empirijskih istraživanja omogućava definisanje kvalitetnih anotacionih smernica [14]	1
Klasa podataka	• često se javlja u kombinaciji sa drugim mirisima [53][108] • potrebno je postići konsenzus oko relevantnih metrika za detekciju [14] • potrebno je empirijsko istraživanje usled manjka postojećih istraživanja [17]	2
Zavist među odlikama	• čini softver sklonim greškama [24][45][107] • često se javlja u kombinaciji sa drugim mirisima [31][106] • potrebno je postići konsenzus oko relevantnih metrika za detekciju [14] • potrebno je empirijsko istraživanje usled manjka postojećih istraživanja [17]	2
Odbačeno nasledstvo	• poprilično rasprostranjen (u 58% analiziranih primera) [24] • čini softver sklonim greškama i izmenama [24] • često se javlja u kombinaciji sa drugim mirisima [31][106] • potrebno je postići konsenzus oko relevantnih metrika za detekciju [14] • potrebno je empirijsko istraživanje usled manjka postojećih istraživanja [17]	2

Odgovaranje na istraživačka pitanja: Na osnovu analize podataka prikupljenih tokom eksperimenata će biti utvrđeno da li projektovano rešenje ispunjava zahteve definisane na osnovu ciljeva interesnih grupa. Ukoliko rešenje ispunjava zahteve, može se zaključiti da pomaže ostvarenju ciljeva interesnih grupa. Na kraju, moguće je odgovoriti na istraživačka pitanja i diskutovati doprinose i ograničenja rešenja. U ovom istraživanju se odgovaranje na istraživačka pitanja svodi na proveru postavljenih hipoteza.

Potpoglavlje 5.4 analizira ispunjenost zahteva definisanih u Tabeli 1.1 u Potpoglavlju 1.3, dok se u Potpoglavlju 5.5 proveravaju postavljene hipoteze. Poglavlje 6 obuhvata diskusiju o svim doprinosima i ograničenjima rešenja.

4. REZULTATI

Rešenje za ručno anotiranje mirisa koda zasnovano na preskriptivnoj anotacionoj paradigmi je glavni rezultat ovog istraživanja. Njen cilj je da podrži anotatore da proizvedu visokokvalitetne skupove podataka mirisa koda. Sa kvalitetnim skupovima podataka je moguće obučiti ML modele za detekciju mirisa i poboljšanje održivosti koda. Kreirano rešenje je rezultat dve iteracije ciklusa dizajniranja, opisanog u Poglavlju 3.

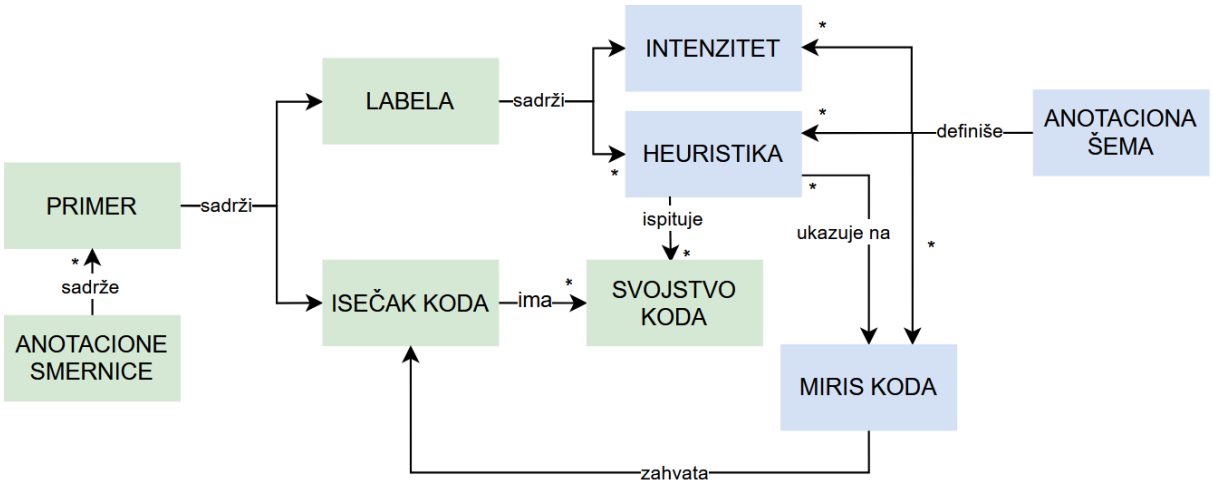
U narednim potpoglavljima će detaljno biti opisano rešenje, koje se sastoji od *anotacione procedure* koja usvaja preskriptivnu paradigmu (Potpoglavlje 4.2) i *alata za anotiranje* koji anotatori koriste tokom anotacione procedure (Potpoglavlje 4.3). Pre ovih potpoglavlja, u Potpoglavlju 4.1 je predstavljen konceptualni model koji omogućava razumevanje koncepata relevantnih za rezultat istraživanja.

Sekundarni rezultati ovog istraživanja su nastali tokom validacije rešenja za ručno anotiranje mirisa koda, te su predstavljeni u Poglavlju 5. Rezultati obuhvataju anotacione šeme i smernice za *dugačku metodu*, *veliku klasu*, *klasu podataka*, *zavist među odlikama i odbačeno nasledstvo* (Potpoglavlje 5.2) i anotirani skup podataka (Potpoglavlje 5.3).

4.1 Koncepti

Konceptualni model je podeljen na dva dela: prvi deo predstavlja koncepte vezane za anotacionu šemu i smernice, a drugi deo koncepte vezane za skup podataka.

Na Slici 4.1 su prikazani koncepti vezani za anotacionu šemu i smernice. U rešenju za ručno anotiranje mirisa koda, *anotaciona šema* definiše sve koncepte koje je potrebno obuhvatiti skupom podataka. Glavni koncept koji se anotira je *miris koda*, koji je detaljno opisan u Potpoglavlju 2.1. Zatim, anotaciona šema definiše *heuristike* (opisane u Potpoglavlju 2.2) koje ukazuju na prisustvo *mirisa koda* u *isečku koda*. Najzad, anotaciona šema definiše *intenzitete* mirisa koda, koje će anotatori dodeljivati prilikom anotiranja.

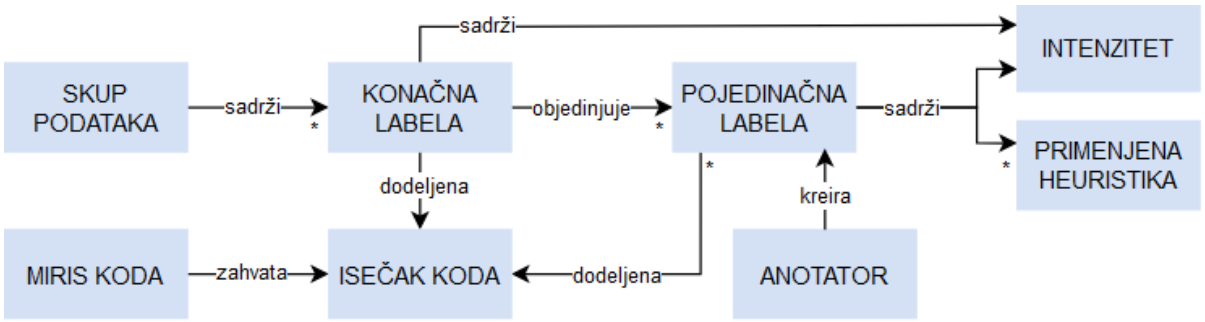


Slika 4.1 Koncepti vezani za anotacionu šemu i anotacione smernice (označeni redom plavom i zelenom).

Anotacione smernice kroz *primere* daju uputstva anotatorima kako se anotaciona šema primenjuje u praksi. *Primer* se sastoji od *isečka koda* kom je dodeljena *labela.* Labela se sastoji od *intenziteta* mirisa koda i *heuristika* koje su primenjive u datom primeru. Pored toga,

smernice definišu *svojstva koda* koja su relevantna za analizu heuristika u datom primeru. Svojstva koda su detaljnije opisana u Potpoglavlju 2.2 i 2.3.

Na Slici 4.2 se nalaze koncepti vezani za skup podataka. *Skup podataka* predstavlja kolekciju *konačnih labela,* koje predstavljaju *intenzitet* mirisa koda koji je dodeljen *isečku koda*. Konačna labela se može koristiti za obučavanje ML modela za detekciju mirisa. Ona je dobijena spajanjem *pojedinačnih labela* koje je svaki *anotator* samostalno dodelio isečku. Intenzitet *konačne labele* je onaj koji je bio najzastupljeniji u pojedinačnim labelama (engl. *majority vote aggregation*). *Pojedinačna labela* se sastoji od *intenziteta* mirisa koda i *primenjenih heuristika*.



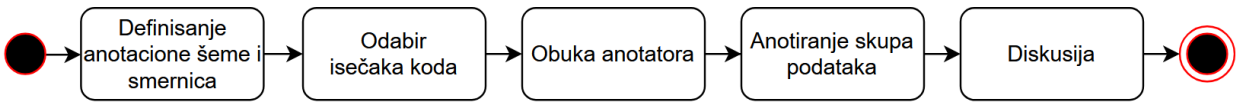
Slika 4.2 Koncepti vezani za skup podataka mirisa koda

4.2 Procedura za ručno anotiranje mirisa koda

Anotaciona procedura je rezultat dve iteracije ciklusa dizajniranja predstavljenog u Poglavlju 3. U prvoj iteraciji je procedura kreirana imajući u vidu „mirise unutar klasa“ koji se mogu analizirati u okviru izolovanog isečka koda [45]. Kako bi se prevazišlo ograničenje procedure koja podrazumeva analizu izolovanih isečaka koda, u drugoj iteraciji su u obzir uzeti “mirisi između klasa”, koji pored analize posmatranog isečka koda zahtevaju i analizu dizajna softverskog sistema [45].

Glavne aktivnosti anotacione procedure prikazane su na Slici 4.3. Tokom prve aktivnosti se na osnovu relevantne literature vrši *definisanje anotacione šeme i smernica* za odabrane mirise koda. Preskriptivna paradigma koju usvaja anotaciona procedura podrazumeva razvoj šeme i smernica kako bi se smanjila subjektivnost anotatora i njihove međusobne nesuglasice, čime se obezbeđuje veća konzistentnost anotacija. Zatim sledi *odabir isečaka koda* koji će se anotirati, pri čemu je važno odabrati aktivne softverske projekte različitih veličina i domena kako bi se obezbedilo kreiranje robustnog skupa podataka. Treća aktivnost u proceduri je *obuka anotatora* u kojoj anotatori stiču potrebno znanje za anotiranje mirisa koda. Obuka smanjuje nesuglasice među anotatorima, a povećava njihovu međusobnu saglasnost i konzistentnost anotacija. Nakon obuke, anotatori prelaze na *anotiranje skupa podataka,* gde stečeno znanje primenjuju kako bi analizirali isečke koda kroz svojstva koda, otkrili mirise putem heuristika i odredili intenzitet mirisa. Konačno, *diskusija* služi za razrešavanje svih konfliktnih anotacija, čime se značajno doprinosi poboljšanju kvaliteta skupa podataka za obučavanje ML modela, povećavajući konzistentnost anotacija.

Sve navedene aktivnosti su detaljno opisane u potpoglavljima koja slede.

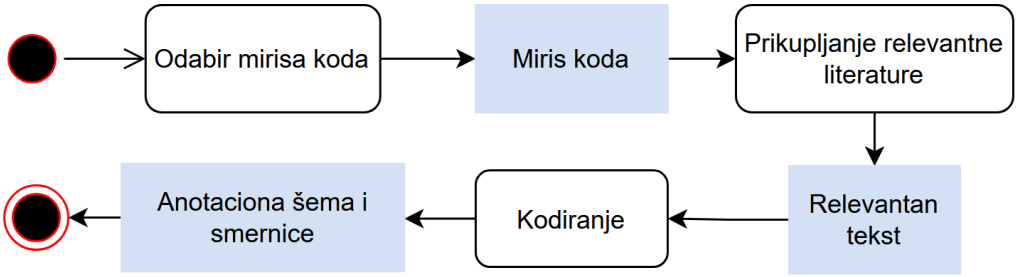


Slika 4.3 Pregled glavnih aktivnosti anotacione procedure

4.2.1 Definisanje anotacione šeme i smernica

Prva aktivnost u anotacionoj proceduri je *definisanje anotacione šeme i smernica*, predstavljena na Slici 4.4 i sadrži sledeće podaktivnosti:

1. *Odabir mirisa koda:* Prilikom odabira mirisa koda se mogu razmotriti različiti aspekti, poput rasprostranjenosti mirisa, sklonosti softvera greškama ili izmenama zbog prisustva mirisa, koegzistencije sa drugim mirisima, složenosti detekcije mirisa i zastupljenosti mirisa u postojećim istraživanjima. Ukoliko je cilj napraviti skup podataka koji sadrži dokazano štetne mirise, tada se biraju mirisi za koje brojna istraživanja pokazuju da su veoma rasprostranjeni, da povećaju broj grešaka i izmena softvera i da se često javljaju sa drugim mirisima. Sa druge strane, cilj može biti kreiranje skupa podataka koji će doprineti istraživačkoj zajednici novim otkrićima, te se tada biraju mirisi koji su složeni za detekciju i koji nisu u velikoj meri zastupljeni u postojećoj literaturi.
2. *Prikupljanje relevantne literature:* Za odabrane mirise koda se prikuplja relevantna literatura, koja uključuje definicije i opise mirisa, iskustva iz industrije, empirijska istraživanja o tome kako softverski inženjeri doživljavaju mirise koda u praksi i istraživanja vezana za automatsku detekciju mirisa koda.
3. *Kodiranje:* Prikupljen relevantan tekst koji sadrži opise, zapažanja inženjera i pravila za detekciju se tehnikom kodiranja preslikava na diskretan skup vrednosti [60]. Tačnije, iz relevantnog teksta se izvlači skup heuristika, svojstava koda i intenziteta. Na osnovu njih se kreiraju anotaciona šema i smernice (na Slici 4.1 se vidi koje koncepte sadrži anotaciona šema, a koje smernice) koje će anotatori koristiti tokom obuke i anotiranja skupa podataka.



Slika 4.4 Podaktivnosti u sklopu definisanja *anotacione šeme i smernica* obuhvataju odabir mirisa koda, prikupljanje relevantne literature i kodiranje.

4.2.2 Odabir isečaka koda

Druga aktivnost anotacione procedure podrazumeva odabir isečaka koda. Isečke treba izabrati nasumično iz odabranih projekata, a zatim ukloniti trivijalne isečke koji, na primer, imaju jednu ili dve linije koda, kako bi se uštedelo vreme anotatora. Tom i saradnici [109] su predložili kriterijume kojima se treba voditi prilikom odabira projekata za detekciju mirisa koda:

- Potrebno je izabrati projekte koji se aktivno razvijaju. Programski jezici se često menjaju, a obrasci koji se koriste u kodu mogu postati zastareli i nebitni [27][110].
- Potrebno je izabrati projekte koji su relativno popularni. Popularnost ukazuje na prihvaćenost i korisnost projekta za zajednicu. Na ovaj način se izbegavaju jednostavni i nerazvijeni projekti koji bi mogli biti obrisani ili napušteni.
- Treba odabrati projekte koji su zreli, pri čemu se broj promena (engl. *commit*) može koristiti za procenu zrelosti projekta [110].
- Odabrani projekti treba da pokrivaju različite stilove kodiranja kako bi se osigurala sveobuhvatna analiza mirisa.

4.2.3 Obuka anotatora

Treća aktivnost u proceduri je *obuka anotatora* tokom koje anotatori stiču neophodno znanje za anotiranje mirisa koda. Obuka je iterativna aktivnost čije se podaktivnosti ponavljaju sve dok se ne postigne usaglašenost anotatora po pitanju razumevanja mirisa koje treba anotirati. Podaktivnosti obuke su prikazane na Slici 4.5 i obuhvataju:

1. *Analiza anotacione šeme i smernica:* Svaki anotator samostalno proučava anotacionu šemu i smernice, kako bi postigao razumevanje o mirisima koje će anotirati, svojstvima koda i heuristikama koje će analizirati tokom detekcije mirisa i intenzitetima koje će dodeljivati otkrivenim mirisima. Pored toga, svaki anotator samostalno proučava smernice kako bi razumeo koja svojstva koda su relevantna za analizu određenih mirisa, kako se svojstva koda koriste za određivanje heuristika i kako se određuje intenzitet na osnovu heuristika.
2. *Probno anotiranje:* Anotatori testiraju stečeno znanje kroz probno anotiranje (engl. *Proof-Of-Concept* – POC) manjeg skupa podataka. Svaki anotator samostalno anotira isečke koda prateći sledeće korake:
    a. *Detekcija mirisa koda:* Anotator analizira isečak koda kroz svojstva koda relevantna za analizu određenog mirisa, što je definisano anotacionim smernicama. Tokom analize svojstava koda, anotator primenjuje heuristike koje anotaciona šema definiše kao relevantne za određivanje prisustva mirisa koda.
    b. *Određivanje intenziteta:* Anotator određuje intenzitet mirisa koda koji je prisutan u analiziranom isečku koda. Skup dostupnih vrednosti intenziteta je definisan anotacionom šemom, dok smernice upućuju anotatora kako da odredi intenzitet na osnovu primenjivih heuristika.
    c. *Kreiranje pojedinačne labele:* Anotator kreira pojedinačnu labelu koja se sastoji od heuristika koje je anotator označio kao primenjive i intenziteta. Pojedinačna labela se odnosi na isečak koda u kom se tražio određeni miris koda.
3. *Diskutovanje konflikata:* Tokom probnog anotiranja, različiti anotatori mogu dodeliti različite intenzitete istim isečcima koda. Ukoliko nije moguće odrediti intenzitet konačne labele[13], neophodno je prodiskutovati neslaganja i ponoviti obuku. Kako bi

[13] Intenzitet konačne labele je onaj koji je bio najzastupljeniji u pojedinačnim labelama.

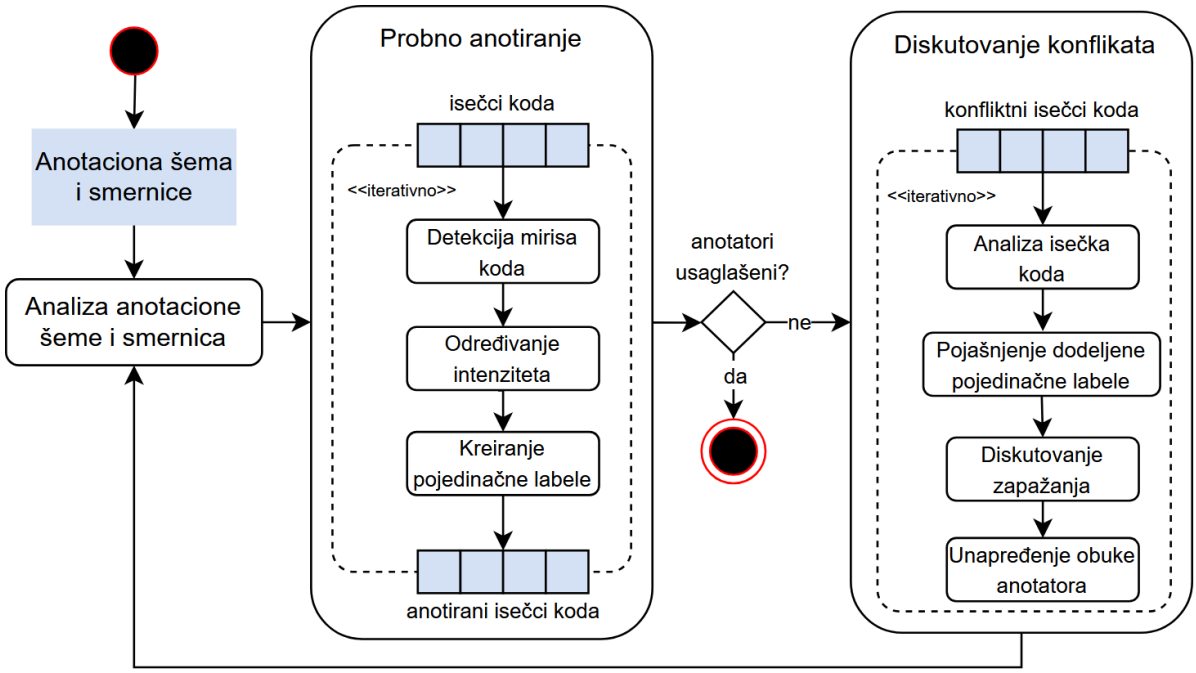
prodiskutovali neslaganja, anotatori prolaze kroz sledeće korake za svaki konfliktan isečak koda:

a. *Analiza isečka koda:* Anotatori analiziraju konfliktni isečak koda kroz svojstva koda, kako bi se pripremili za detaljnu diskusiju o nesuglasicama.

b. *Pojašnjenje dodeljene pojedinačne labele:* Anotatori obrazlažu dodeljeni intenzitet i heuristike koje su označili kao primenjive.

c. *Diskutovanje zapažanja:* Anotatori diskutuju greške i previde koje su možda napravili tokom anotiranja. Zatim diskutuju neočekivane primere koje anotaciona šema i smernice nisu pokrile, a koji su doveli do konflikata.

d. *Unapređenje obuke anotatora:* Anotaciona šema i smernice se unapređuju na osnovu prodiskutovanih zapažanja.



Slika 4.5 Obuka anotatora obuhvata analizu anotacione šeme i smernica, probno anotiranje i diskutovanje konflikata.

4.2.4 Anotiranje skupa podataka

Nakon obuke, anotatori započinju aktivnost *anotiranja skupa podataka*. Podaktivnosti koje anotiranje podrazumeva se mogu videti u narandžastoj sekciji na Slici 4.6:

1. *Anotiranje:* Ova aktivnost je veoma slična probnom anotiranju u okviru obuke anotatora, međutim ima drugačiji cilj. Cilj probnog anotiranja je da anotatori testiraju stečeno znanje i usaglase razumevanje mirisa. Probni skup podataka je relativno mali i svaki isečak koda je anotiran od strane svih anotatora kako bi se mogle diskutovati nesuglasice, a šema i smernice unaprediti. Sa druge strane, aktivnost anotiranja skupa podataka ima za cilj izgradnju kvalitetnog skupa podataka i podrazumeva veći broj isečaka koda. Ručno anotiranje je vremenski zahtevno, pa su isečci koda podeljeni među anotatorima tako da svaki isečak prvo anotiraju samo dva anotatora, a ne svi.

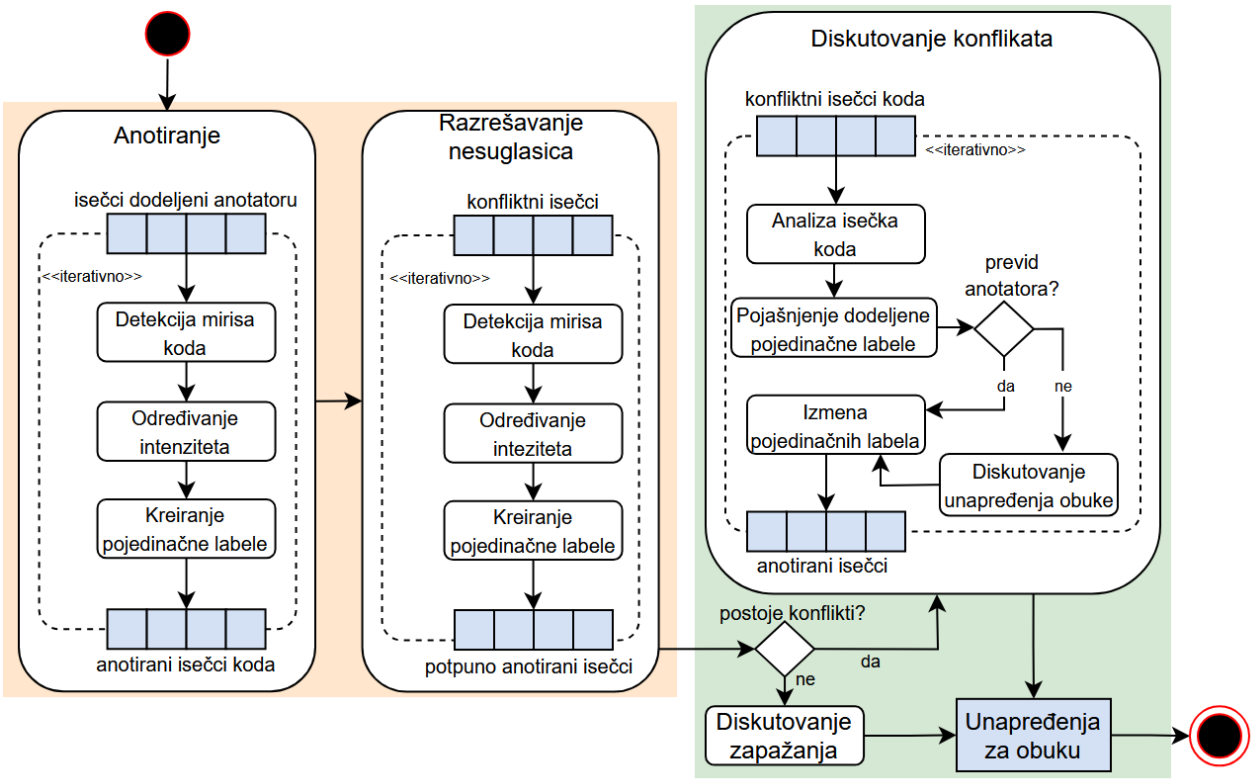
Ovakvom distribucijom isečaka među anotatorima je moguće napraviti veće skupove podataka.

2. *Razrešavanje nesuglasica:* Pojedinačne labele anotatora sadrže intenzitete na osnovu kojih se formira konačna labela za svaki isečak koda. Intenzitet konačne labele je onaj koji je bio najzastupljeniji u pojedinačnim labelama. Ukoliko nije moguće odrediti konačnu labelu zato što su dva anotatora dodelila različite intenzitete, isečak koda će biti anotiran od strane ostalih anotatora koji ga prvobitno nisu anotirali. Ovi anotatori nemaju uvid u prethodno dodeljene anotacije. Na ovaj način je moguće razrešiti konflikte prva dva anotatora. Međutim, ukoliko nije moguće razrešiti konflikte nakon dodatnog anotiranja ostalih anotatora, isečak koda će biti analiziran u narednoj aktivnosti anotacione procedure.

4.2.5 Diskusija

Poslednja aktivnost u proceduri je *diskusija,* tokom koje se razrešavaju sve nesuglasice i, po potrebi, unapređuje celokupna anotaciona procedura. Podaktivnosti diskusije se mogu videti u zelenoj sekciji na Slici 4.6:

1. *Diskutovanje konflikata:* U ovoj podaktivnosti anotatori treba da prodiskutuju konfliktne anotacije nastale tokom prethodne aktivnosti. Za svaki isečak koda anotatori prolaze kroz sledeće korake:
    a. *Analiza isečka koda*: Anotatori analiziraju konfliktni isečak koda kroz svojstva koda, kako bi se pripremili za detaljnu diskusiju.
    b. *Pojašnjenje dodeljene pojedinačne labele:* Anotatori obrazlažu dodeljeni intenzitet i heuristike koje su označili kao primenjive.
    c. *Izmena pojedinačnih labela*: Nakon diskusije o posmatranom isečku koda, anotatori menjaju pojedinačne labele dok ne postignu konačan konsenzus.
    d. *Diskutovanje unapređenja obuke:* Ako su konfliktne anotacije nastale zbog propusta u anotacionoj šemi ili smernicama, razmatraju se moguća unapređenja obuke anotatora (smernica, šeme i probnog anotiranja) koja će sprečiti slične konflikte u budućnosti.
2. *Diskutovanje zapažanja:* Nakon rešenih konflikata ili ako nakon anotiranja skupa podataka ne postoje konfliktne anotacije koje bi se prodiskutovale, u ovoj aktivnosti anotatori mogu podeliti bilo koja zapažanja koja su imali tokom anotiranja, a koja bi mogla unaprediti celokupnu anotacionu proceduru.



Slika 4.6 Anotacija i *diskusija* (naglašene redom narandžastim i zelenim delom dijagrama).

4.3 Alat za anotiranje – *DataSet Explorer*

Drugi deo rešenja za ručno anotiranje mirisa koda je alat *DataSet Explorer* (DSE). DSE alat je razvijen kako bi ubrzao i olakšao rad anotatora tokom preskriptivne anotacione procedure, čime se povećava kvalitet rezultujućih skupova podataka koji se koriste za obučavanje ML modela za detekciju mirisa. Kao i anotaciona procedura, alat je rezultat dve iteracije ciklusa dizajniranja opisanog u Poglavlju 3. U prvoj iteraciji je alat dizajniran i implementiran kako bi omogućio anotatorima anotiranje „mirisa unutar klasa“ kroz analizu izolovanog isečka koda [45]. U drugoj iteraciji je alat unapređen kako bi omogućio i analizu dizajna sistema što je neophodno za anotiranje “mirisa između klasa” [45].

DSE alat pruža podršku anotatorima tokom preskriptivne anotacione procedure, jer nudi niz funkcionalnosti koje olakšavaju njihov rad u svakoj aktivnosti procedure. Za prvu aktivnost, *definisanje anotacione šeme i smernica*, alat omogućava kreiranje anotacione šeme za bilo koji Fowlerov miris koda [8]. Zatim, aktivnost *odabir isečaka koda* podržana je funkcionalnostima kao što su kloniranje repozitorijuma sa GitHub-a, nasumičan odabir isečaka i uklanjanje trivijalnih isečaka.

Za treću aktivnost, *obuku anotatora*, alat omogućava analizu anotacione šeme i probno anotiranje. Alat će označiti sve konfliktne anotacije i omogućiti anotatorima diskusiju o njima. Četvrta aktivnost podrazumeva *anotiranje skupa podataka*, pri čemu alat omogućava analizu C# isečaka koda kroz automatsko izvlačenje i vizuelizaciju više svojstava koda: izvorni kod, strukturne metrike i veze među komponentama sistema. Anotatori kreiraju pojedinačne labele beležeći primenjene heuristike i intenzitete u anotacionoj formi. Alat formira konačnu labelu na osnovu pojedinačnih ukoliko je to moguće. Alat izdvaja konfliktne isečke koda za koje nije

moguće formirati konačnu labelu, kako bi ih anotirali anotatori koji ih nisu prvobitno anotirali. Konačno, u aktivnosti *diskusije*, alat izdvaja isečke za koje nije postignut konsenzus, kako bi anotatori mogli da ih analiziraju, diskutuju i reše konflikte. Funkcionalnosti su detaljnije opisane u Potpoglavlju 4.3.2.

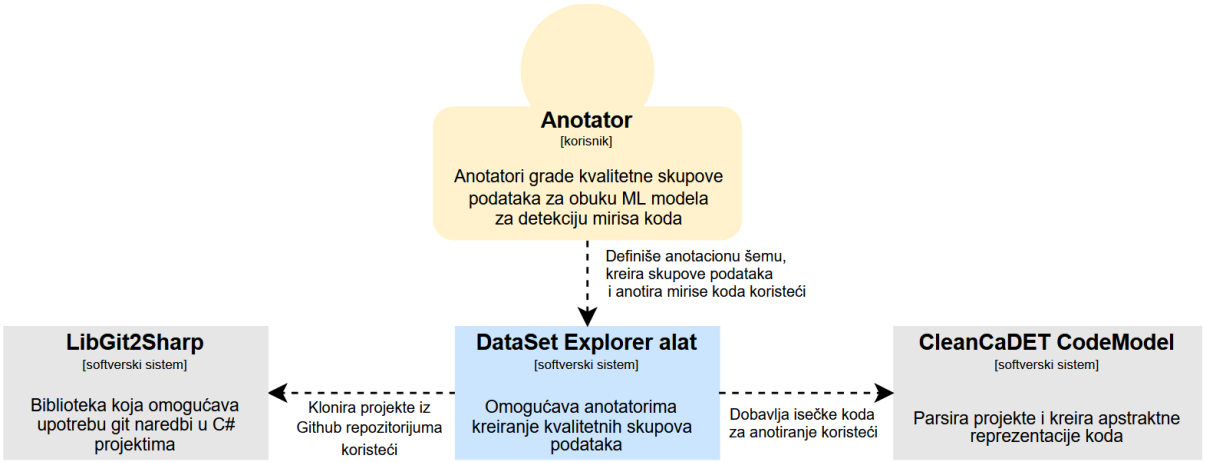
DataSet Explorer alat je dostupan javnosti, a u Tabeli 4.1 se nalaze korisni resursi za pristup alatu.

Tabela 4.1 Korisni resursi za pristup i korišćenje *DataSet Explorer* alata.

Resurs	Opis	Link
Back-end izvorni kod	Repozitorijum koji sadrži izvorni kod serverske aplikacije.	https://github.com/Clean-CaDET/dataset-explorer
Front-end izvorni kod	Repozitorijum koji sadrži izvorni kod korisničkog interfejsa.	https://github.com/Clean-CaDET/platform-explorer-ui-web
Dokumentacija	Kolekcija *wiki* stranica koja opisuje dizajn i funkcionalnosti alata.	https://github.com/Clean-CaDET/dataset-explorer/wiki

4.3.1 Arhitektura alata

Dijagam na Slici 4.7 prikazuje interakciju na visokom nivou između *DataSet Explorer* alata, njegovih korisnika i drugih sistema. Glavni korisnici DSE alata su *anotatori,* čiji je cilj izgradnja skupova podataka visokog kvaliteta koji su korisni za obučavanje ML modela. *DSE alat* koristi biblioteku *LibGit2Sharp*[14] za kloniranje repozitorijuma sa GitHub-a. Zatim, *DSE alat* koristi *CleanCaDET CodeModel*[15] za parsiranje kloniranih repozitorijuma u C# isečke koda koje anotatori analiziraju i anotiraju.



Slika 4.7 Interakcija DSE alata sa korisnicima i drugim sistemima.

Na Slici 4.8 prikazan je šema baze podataka u okviru DSE alata, koja prikazuje najvažnije klase i veze među njima, povezane sa konceptom skupa podataka. Skup podataka je centralni koncept alata, jer je alat razvijen kako bi omogućio anotatorima kreiranje kvalitetnih skupova podataka.

[14] LibGit2Sharp biblioteka je dostupna na sledećem linku https://github.com/libgit2/libgit2sharp.

[15] *CleanCaDET CodeModel* je modul *CleanCaDET* platforme i dostupan je na sledećem linku https://github.com/Clean-CaDET/platform. *CodeModel* modul je servis koji podržava funkcionalnosti DSE alata i predstavlja apstraktnu reprezentaciju koda.

Šema baze podataka pruža uvid u strukturu i organizaciju podataka koji se koriste unutar DSE alata, čime pomaže u razumevanju načina na koji alat funkcioniše.

Svaki skup podataka (*Dataset*) se sastoji od projekata (*DatasetProject*) i kreira se za jedan ili više mirisa koda (*CodeSmell*). Projekat sadrži isečke koda (*Instance*) koji su grupisani u kandidatske grupe (*SmellCandidateInstance*). Kandidatska grupa se odnosi na jedan miris koda i sadrži isečke koda u kojima je anotiran jedan specifičan miris koda. Na primer, kreiran je skup podataka za dva mirisa: *dugačka metoda* i *velika klasa*. Svaki projekat unutar skupa podataka sadrži po dve kandidatske grupe: jednu za isečke koda u kojima se anotira *dugačka metoda*, a drugu za isečke u kojima se anotira *velika klasa*.

Svaki isečak koda (*Instance*) ima povezane isečke (*RelatedInstance*), što je od značaja za detaljnu analizu dizajna i semantike sistema. Isečci mogu biti povezani na različite načine, tj. imati različite tipove veza (*RelationType*):

1. Jedan isečak može da referencira drugi isečak (*References*)
2. Jedan isečak može biti referenciran od strane drugog isečka (*Referenced*)
3. Jedan isečak može predstavljati roditeljsku klasu klase u drugom isečku (*Parent*)
4. Jedan isečak može predstavljati klasu naslednicu klase u drugom isečku (*Subclass*)
5. Jedan isečak može pripadati drugom isečku koda (*BelongsTo*)

Pored tipa veze (*RelationType*), postoji i tip spregnutosti (*CouplingType*) koji preciznije određuje na koji način su dva isečka koda povezana. Tipovi veze *Parent*, *Subclass* i *BelongsTo* imaju isti tip spregnutosti, dok tipovi veze *References* i *Referenced* mogu imati sledeće tipove spregnutosti:

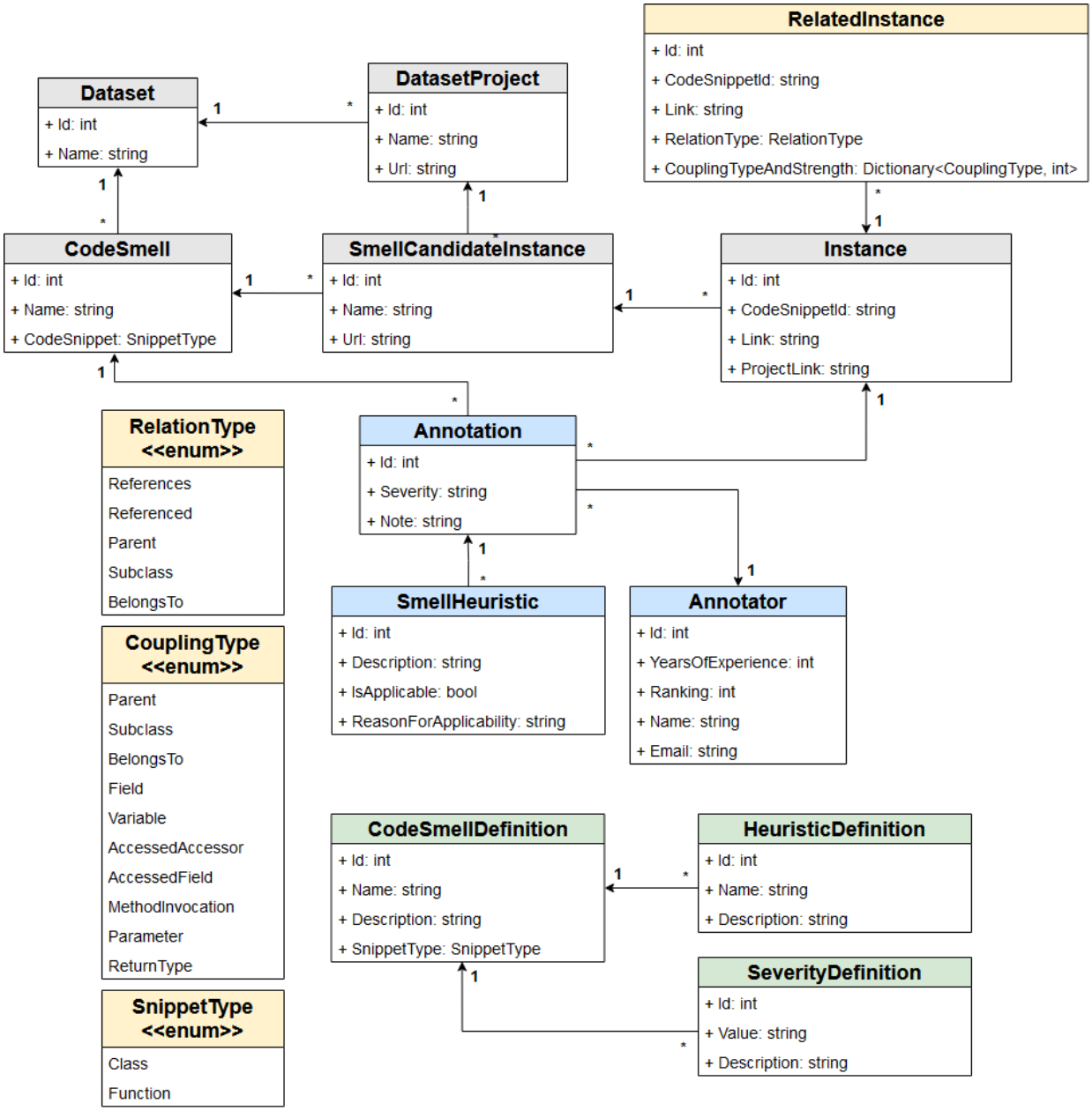
1. *Field* označava da jedna klasa sadrži polje koje je tipa druge klase[16].
2. *Variable* označava da se u okviru jedne klase nalazi promenljiva koja je tipa druge klase.
3. *AccessedAccessor* označava da se u okviru jedne klase pristupa *get* ili *set* metodama druge klase.
4. *AccessedField* označava da se u okviru jedne klase pristupa poljima druge klase.
5. *MethodInvocation* označava da se u okviru jedne klase pozivaju metode druge klase.
6. *Parameter* označava da u jednoj klasi postoji parametar metode koji je tipa druge klase.
7. *ReturnType* označava da u jednoj klasi postoji povratna vrednost metode koja je tipa druge klase.

Svaki isečak koda (*Instance*) sadrži više anotacija (*Annotation*), pri čemu je svaku anotaciju pravi jedan anotator (*Annotator*). Svaka anotacija sadrži heuristike (*SmellHeuristic*). Na primer, prilikom anotiranja *dugačke metode*, anotatori treba da odrede da li su sledeće heuristike primenjive: *metoda je dugačka*, *metoda je kompleksna* i *metoda ima više odgovornosti*. Anotacija će sadržati ove tri heuristike, pri čemu će anotatori označiti koje od njih su primenjive na posmatrani isečak koda.

Heuristike koje anotatori analiziraju prilikom detekcije mirisa i intenziteti mirisa koje dodeljuju (*Severity* unutar *Annotation*) su određeni anotacionom šemom. Anotaciona šema sastoji se od

[16] Isečak može predstavljati funkciju ili klasu (*SnippetType*). Radi preciznosti opisa, u tipovima spregnutosti (*CouplingType*) su navedene klase, a ne isečci koda.

definicija mirisa (*CodeSmellDefinition*), pri čemu je za svaki miris definisano više heuristika (*HeuristicDefinition*) i intenziteta (*SeverityDefinition*).



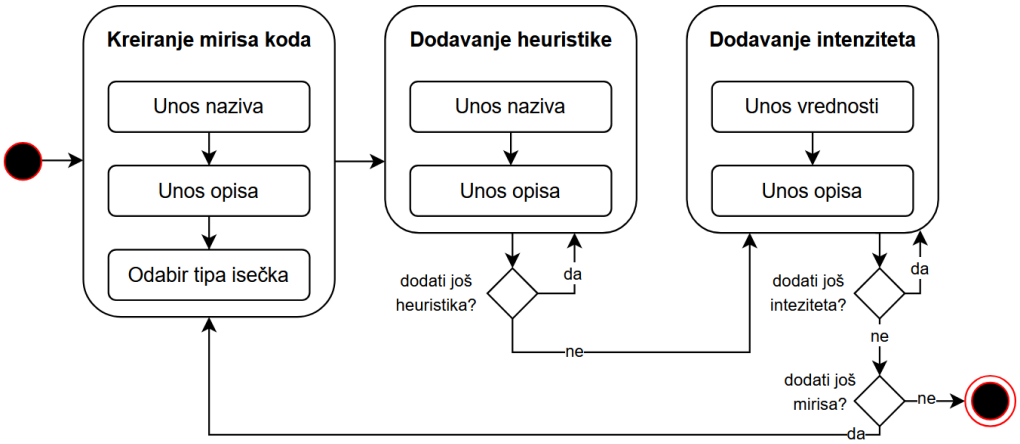
Slika 4.8 Šema baze podataka u DSE alatu. Prikazane su najvažnije klase i veze među njima vezane za koncept skupa podataka. Klase označene sivom su vezane za organizaciju skupa podataka. Klase označene žutom su relevantne za analizu povezanih isečaka koda. Plave klase se odnose na anotacije, a zelene klase na anotacionu šemu.

4.3.2 Funkcionalnosti alata

DSE alat nudi korisnicima funkcionalnosti za upravljanje anotacionom šemom, kreiranje skupova podataka, anotiranje skupova podataka i diskutovanje anotacija. U nastavku će biti

opisane najvažnije funkcionalnosti. Video materijali koji demonstriraju ove funkcionalnosti su dostupni javnosti[17].

DSE alat omogućava definisanje anotacione šeme putem upravljanja mirisima koda i njihovim heuristikama i intenzitetima. Heuristike definisane za određeni miris koda će biti dostupne tokom anotiranja isečaka, a anotator će svaku od njih razmotriti prilikom detekcije mirisa. Takođe, anotatori će moći da izaberu jedan od prethodno definisanih intenziteta. Korisnik može kreirati, pregledati, ažurirati i obrisati mirise, heuristike i intenzitete[18]. Na Slici 4.9 je prikazan dijagram aktivnosti koji ilustruje funkcionalnost kreiranja pomenutih koncepata unutar anotacione šeme.



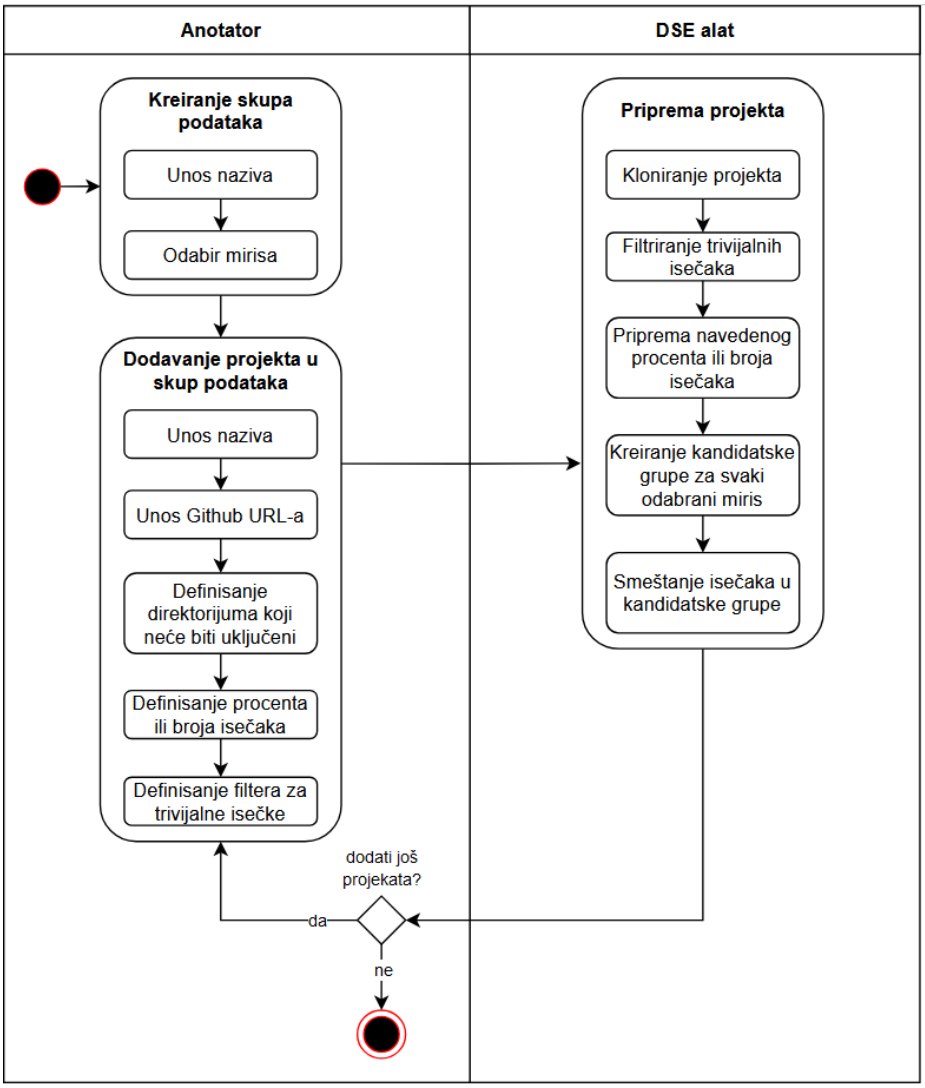
Slika 4.9 Dijagram aktivnosti za definisanje anotacione šeme.

Korisnici DSE alata mogu kreirati, pregledati, ažurirati i obrisati skupove podataka. Skupovi podataka su inicijalno prazni, što znači da ne sadrže projekte i nisu anotirani. Korisnik može dodati projekte u skup podataka, pregledati ih, ažurirati i obrisati[19]. Nakon dodavanja projekata u skup podataka, sistem priprema projekte kako bi anotatori mogli započeti anotiranje. Dijagram aktivnosti na Slici 4.10 ilustruje kreiranje skupa podataka, dodavanje projekata i pripremu projekata od strane sistema.

[17] Video materijali na kojima su prikazane funkcionalnosti DSE alata su dostupni na https://www.youtube.com/playlist?list=PLOXMk-hlotHGGoyNgCEk285CQkUEUNCAW.

[18] Video koji prikazuje funkcionalnosti vezane za anotacionu šemu je dostupan na https://www.youtube.com/watch?v=Tzc0Gv4c7oI.

[19] Video koji prikazuje funkcionalnosti vezane za upravljanje skupovima podataka i projektima je dostupan na https://www.youtube.com/watch?v=WFFYSdDnyR4.

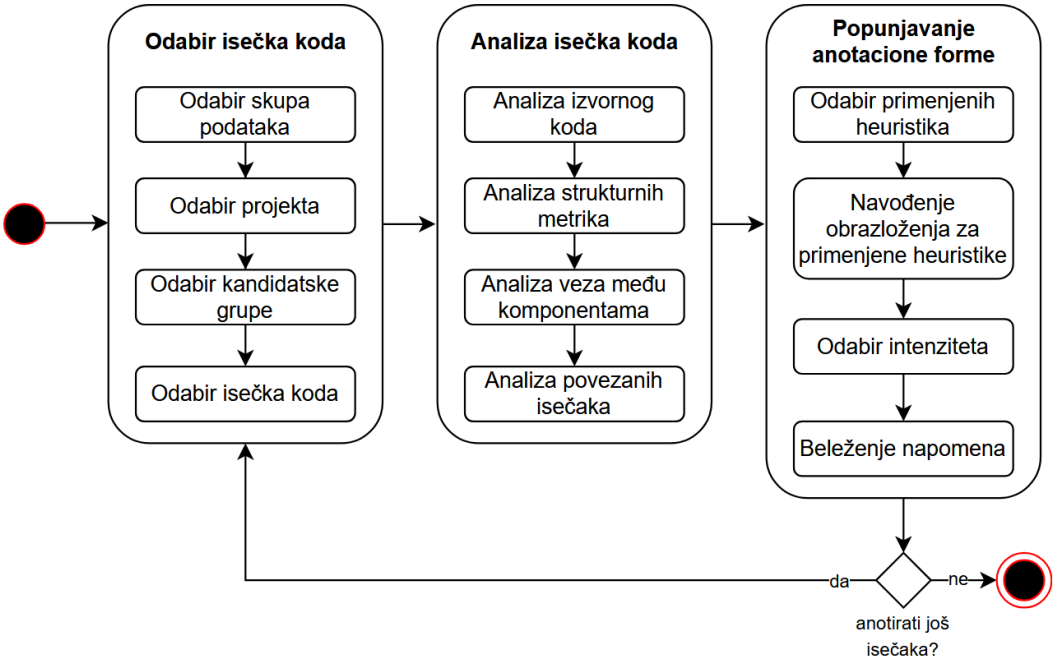


Slika 4.10 Dijagram aktivnosti za kreiranje skupa podataka i dodavanje projekata.

Nakon kreiranja skupa podataka i dodavanja projekata, anotatori mogu započeti anotiranje. Anotator pristupa isečku koda tako što bira skup podataka, projekat, kandidatsku grupu i isečak koda. Kandidatska grupa definiše koji miris koda se anotira. Za odabrani isečak koda, DSE alat omogućava analizu svojstava koda koja uključuju izvorni kod, strukturne metrike i veze među komponentama. Nakon analize svojstava, anotator popunjava anotacionu formu, gde označava heuristike koje su primenjive i obrazlaže svoje rezonovanje. Zatim označava intenzitet mirisa. Anotaciona forma je usklađena sa anotacionom šemom, tj., prikazuje heuristike i intenzitete koji su definisani za miris koda koji se anotira. Anotatori mogu pregledati i izmeniti svoje anotacije. Dijagram aktivnosti vezanih za kreiranje anotacije se nalazi na Slici 4.11[20]. Opisana aktivnost anotiranja se može ubrzati pomoću DSE alata, koji nudi mogućnost automatskog izlistavanja sledećeg isečka koda nakon anotiranja posmatranog isečka[21].

[20] Video koji prikazuje funkcionalnost anotiranja isečka koda se nalazi na https://www.youtube.com/watch?v=68AQ47jwXLE.

[21] Video koji prikazuje automatski odabir sledećeg isečka koda za anotiranje se nalazi na https://www.youtube.com/watch?v=HSfDEgjBvE0.



Slika 4.11 Dijagram aktivnosti za anotiranje isečaka koda. Prilikom analize isečka koda nije neophodno analizirati sva nabrojana svojstva, već one koje su anotacionim smernicama definisane kao relevantne za analizu određenog mirisa koda. Na dijagramu su prikazana sva svojstva radi jednostavnosti prikaza.

Kako bi svaki isečak koda bio anotiran od strane više anotatora i kako bi se razrešile konfliktne anotacije, DSE alat omogućava anotatorima prikaz isečaka koji zahtevaju dodatnu anotaciju. To mogu biti sledeći isečci:

1. Isečci koda koji nisu još uvek anotirani
2. Isečci koda koji su anotirani samo od strane jednog anotatora
3. Isečci koda koji su anotirani od strane više anotatora, ali su njihove anotacije u konfliktu te nije moguće formirati konačnu labelu

Za isečke čije su anotacije u konfliktu, anotatori mogu pregledati popunjene anotacione forme, te izmeniti anotacije nakon diskusije kako bi se postigao konsenzus[22].

Za kraj, DSE alat može generisati različite izveštaje o anotiranom skupu podataka[23].

[22] Video koji prikazuje funkcionalnosti vezane za prikaz isečaka koji zahtevaju dodatnu anotaciju je dostupan na https://www.youtube.com/watch?v=sf9thIzGXHI.

[23] Video koji prikazuje generisanje izveštaja dostupan je na https://www.youtube.com/watch?v=jsaJ5wVWtO4.

5. VALIDACIJA REZULTATA

Validacija rešenja je treća faza ciklusa dizajniranja opisanog u Poglavlju 3. Za potrebe validacije rešenja se može dizajnirati i sprovesti eksperiment u kom se rešenje primenjuje tokom anotiranja skupa podataka. Validacijom rešenja treba pokazati da rešenje ispunjava definisane zahteve (Tabela 1.1). Rešenje koje ispunjava zahteve doprinosi ostvarenju ciljeva istraživanja koji su poravnati sa ciljevima interesnih grupa. Za kraj, potrebno je potvrditi postavljene hipoteze (Poglavlje 1.3).

Rešenje za ručno anotiranje mirisa koda zasnovano na preskriptivnoj anotacionoj paradigmi je validirano u eksperimentu sprovedenom između 2021. i 2023. godine. Tri anotatora su učestvovala u eksperimentu, tokom kog su primenili rešenje, odnosno pratili anotacionu proceduru i koristili DSE alat za anotiranje skupa podataka. Anotatori u okviru grupe imaju različita iskustva. U vreme sprovođenja eksperimenta je prvi anotator bio profesor na četiri predmeta iz oblasti softverskog inženjerstva, sa šest godina iskustva u industriji. Preostali anotatori su studenti doktorskih studija koji se bave istraživanjem održivosti softvera i imaju dve godine iskustva na manjim projektima.

Prve dve aktivnosti anotacione procedure su definisanje anotacione šeme i smernica i odabir isečaka koda. Rezultati ove dve aktivnosti opisani su u Potpoglavljima 5.1 i 5.2. Nakon toga su usledile obuka anotatora, anotiranje skupa i diskusija, te je rezultujući skup podataka opisan u Potpoglavlju 5.3. Na osnovu sprovedenog eksperimenta, u Potpoglavlju 5.4 se analizira ispunjenost zahteva definisanih u Tabeli 1.1. Na kraju, Potpoglavlje 5.5 analizira da li su hipoteze postavljene u Poglavlju 1.3 potvrđene.

5.1 Odabir isečaka koda

Validacija rešenja je izvršena u dve faze. U prvoj fazi su anotatori primenili rešenje kako bi anotirali „mirise unutar klasa“ [45]: *dugačku metodu* i *veliku klasu*. Mirisi ove kategorije su odabrani za prvu fazu, jer se mogu analizirati u izolovanom isečku koda, te ih je jednostavnije i vremenski manje zahtevno anotirati. U drugoj fazi je ista grupa anotatora anotirala „mirise između klasa“ [45]: *klasa podataka*, *zavist među odlikama* i *odbačeno nasledstvo*. Mirise ove kategorije je složenije anotirati pošto zahtevaju analizu dizajna i semantike sistema, te je njihovo anotiranje i vremenski zahtevno. Odabir konkretnih mirisa koda iz ovih kategorija je obrazložen u Potpoglavlju 2.1.1.

Anotatori su u obe faze anotirali isečke koda napisane u C# programskom jeziku. Isečci koda su preuzeti iz projekata koji su javno dostupni na *GitHub*-u. Projekti su birani tako da se obuhvate različite vrste softvera, poput video igara, aplikacija za upravljanje datotekama, i različitih biblioteka. U Tabeli 5.1 se nalazi spisak projekata i broj isečaka iz svakog projekta anotiranog u prvoj fazi validacije za mirise *dugačka metoda* i *velika klasa*. Iz navedenih projekata su pomoću DSE alata uklonjeni isečci koda koji služe za testiranje, pošto istraživanje nije fokusirano na testne mirise koda. Zatim je nasumično odabrano 10% isečaka za anotiranje *dugačke metode* i 10% isečaka za *veliku klasu.* Nakon zapažanja da trivijalni isečci koda ne sadrže mirise, DSE alat je za miris *velika klasa* iz projekata *Core2D*, *Jellyfin*, *MonoGame* i *osu!* uklonio isečke koji imaju manje od pet linija koda i atributa i manje od tri metode. Za miris *dugačka metoda* su uklonjeni isečci koji imaju manje od pet linija koda u svim projektima sem *BurningKnight* koji je prvi anotiran.

Tabela 5.1 Projekti anotirani u prvoj fazi validacije za mirise *dugačka metoda* i *velika klasa*

Naziv	Verzija projekta	Tip softvera	Broj isečaka	
			Velika klasa	Dugačka metoda
BurningKnight	https://github.com/egordorichev/BurningKnight/tree/a55594c11ab681087356af2c129c2d493eba4bd2	Video igra	130	408
ShareX	https://github.com/ShareX/ShareX/tree/c9a71ed00eda0e7c5a45237b9bcd3f8f614cda63	Biblioteka za snimanje ekrana i deljenje sadržaja	81	194
OpenRA	https://github.com/OpenRA/OpenRA/tree/920d00bbae9fa8e62387bbff705ca4bea6a26677	Sistem za razvoj strateških igara	219	441
ShopifySharp	https://github.com/nozzlegear/ShopifySharp/tree/a95f4e3b20dd5d14a1225a48aa2b7e8b3cb15547	Biblioteka za razvoj aplikacija	25	20
Core2D	https://github.com/wieslawsoltes/Core2D/tree/ca290ba82ac571439640af4cb248b2c1e42091d0	Editor za 2D dijagrame	30	120
Jellyfin	https://github.com/jellyfin/jellyfin/tree/6c2eb5fc7e872a29b4a0951849681ae0764dbb8e	Upravljanje sadržajem i emitovanje sadržaja	129	519
MonoGame	https://github.com/MonoGame/MonoGame/tree/4802d00db04dc7aa5fe07cd2d908f9a4b090a4fd	Radni okvir za razvoj video igara	70	279
osu!	https://github.com/ppy/osu/tree/2cac373365309a40474943f55c56159ed8f9433c	Video igra	236	593
Ukupno			920	2574

U Tabeli 5.2 se nalazi spisak projekata i broj isečaka iz druge faze validacije za mirise *klasa podataka*, *zavist među odlikama* i *odbačeno nasledstvo*. Pošto je anotiranje ovih mirisa vremenski zahtevnije od anotiranja mirisa iz prve faze validacije, doneta je odluka da se za svaki miris anotira bar 200 isečaka koda, a ne 10% iz svakog odabranog projekta. Kao i u prvoj fazi, uklonjeni su isečci koda koji služe za testiranje. Pored toga, za svaki miris koda su primenjeni odgovarajući filteri kako bi se uklonili trivijalni isečci koda.

Prilikom anotiranja *klase podataka* nisu uzete u obzir privatne i apstraktne klase. Privatne klase neće biti korišćene od strane drugih klasa u sistemu, a dizajn apstraktnih klasa je opravdan pošto se koriste za centralizovano čuvanje podataka i definisanje metoda koje će koristiti klase naslednice u hijerarhiji. Nad preostalim isečcima je primenjen filter zasnovan na strukturnim metrikama, te su za anotiranje izdvojeni samo oni za koje važi **`WOC < 0.33 V NOPA > 1 V NOPP > 1`**. Ovaj filter izdvaja isečke kao vredne anotiranja ukoliko imaju veći broj članova koji ne implementiraju logiku[24] ili imaju makar dva javna polja (engl. *Number Of Public Attributes* – NOPA) ili dva javna svojstva (engl. *Number Of Public Properties* - NOPP).

Za anotiranje *zavisti među odlikama* su izdvojeni isečci koda koji bar 4 puta pristupaju spoljašnjim podacima (engl. *Access of Import Data* - AID). Isečak koda se smatra suviše trivijalnim ukoliko pristupa spoljašnjim podacima samo nekoliko puta (**`AID < 4`**).

Prilikom anotiranja *odbačenog nasledstva* nisu uzete u obzir apstraktne klase, klase koje nemaju roditeljsku klasu i klase koje nemaju metode ili imaju samo konstruktore. Zatim je primenjen filter zasnovan na strukturnim metrikama **`BUR < 0.33`**. Ovaj filter izdvaja isečke kao vredne anotiranja ukoliko imaju mali procenat upotrebljenih nasleđenih članova[25].

[24] Strukturna metrika WOC (engl. *Weight Of Class*) predstavlja odnos metoda koje implementiraju logiku i članova koji ne implementiraju logiku (javna polja, *get* i *set* metode).

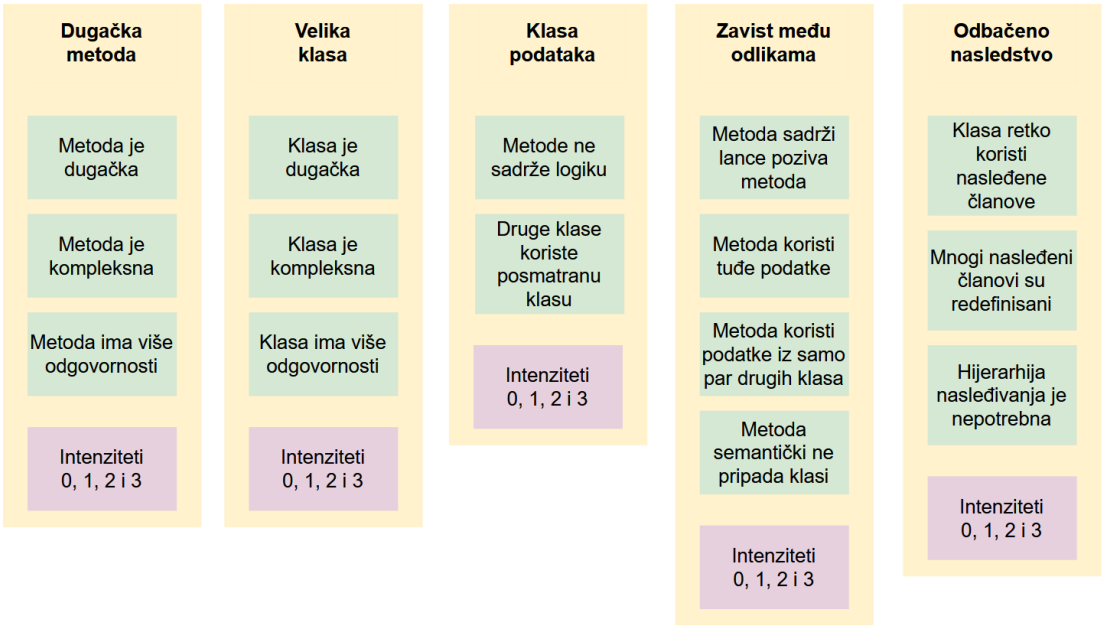
[25] Strukturna metrika BUR (engl. *Base-class Usage Ratio*) računa odnos upotrebljenih nasleđenih članova i ukupnog broja nasleđenih članova.

Tabela 5.2 Projekti anotirani u drugoj fazi validacije za mirise *klasa podataka*, *zavist među odlikama* i *odbačeno nasledstvo*

Naziv	Verzija projekta	Tip softvera	Broj isečaka		
			Klasa podataka	Zavist među odlikama	Odbačeno nasledstvo
BurningKnight	https://github.com/egordorichev/BurningKnight/tree/a55594c11ab681087356af2c129c2d493eba4bd2	Video igra	18	/	/
ShareX	https://github.com/ShareX/ShareX/tree/c9a71ed00eda0e7c5a45237b9bcd3f8f614cda63	Biblioteka za snimanje ekrana i deljenje sadržaja	9	20	20
OpenRA	https://github.com/OpenRA/OpenRA/tree/920d00bbae9fa8e62387bbff705ca4bea6a26677	Sistem za razvoj strateških igara	59	/	/
Jellyfin	https://github.com/jellyfin/jellyfin/tree/6c2eb5fc7e872a29b4a0951849681ae0764dbb8e	Upravljanje sadržajem i emitovanje sadržaja	4	20	20
osu!	https://github.com/ppy/osu/tree/2cac373365309a40474943f55c56159ed8f9433c	Video igra	108	20	20
duplicati	https://github.com/duplicati/duplicati/tree/0a1b32e1887c98c6034c9fafdfddcb8f8f31e207	Servis za čuvanje podataka u oblaku	1	20	20
Sonarr	https://github.com/Sonarr/Sonarr/tree/6378e7afef6072eae20f6408818c6fb1c85661b7	Upravljanje sadržajem sa *Usenet*-a i *BitTorrent*-a	1	20	14
ml-agents	https://github.com/Unity-Technologies/ml-agents/tree/cbc1993c6235cdd033754f9659561e840d4b8708	Softver za obučavanje inteligentnih agenata	4	20	5
Files	https://github.com/files-community/Files/tree/89c33841813a5590e6bf44fb02bb7d06348320c3	Upravljanje datotekama	6	20	4
ScreenToGif	https://github.com/NickeManarin/ScreenToGif/tree/2d318f837946f730e1b2e5c708ae9f776b9e360b	Snimanje ekrana i uređivanje sadržaja različitog formata	5	20	17
Radarr	https://github.com/Radarr/Radarr/tree/5ce1829709e7e1de3953c04e5dab4f3a9d450b38	Upravljanje kolekcijama filmova sa *Usenet*-a i *BitTorrent*-a	2	20	7
Jackett	https://github.com/Jackett/Jackett/tree/db695e5dc01755ff52b5cd7b4f0004ff1035649d	Proxy server	4	/	14
Ryujinx	https://github.com/Ryubing/Ryujinx/tree/81e9b86cdb4b2a01cc41b8e8a4dff2c9e3c13843	Emulator za *Nintendo Switch*	7	/	20
mRemoteNG	https://github.com/mRemoteNG/mRemoteNG/tree/e6d2c9791d7a5e55630c987a3c81fb287032752b	Upravljanje daljinskim vezama	2	/	8
LiteDB	https://github.com/mbdavid/LiteDB/tree/00d28bfafe3c685ae239f759f812def495278eaf	Baza podataka	1	/	/
gitextensions	https://github.com/gitextensions/gitextensions/tree/a866c36b3948dbdddc5e62ade639edc79603a81d	Alat za upravljanje *git* repozitorijumima	/	20	20
Newtonsoft.Json	https://github.com/JamesNK/Newtonsoft.Json/tree/52e257ee57899296d81a868b32300f0b3cfeacbe	Biblioteka za upravljanje JSON podacima	/	20	9
ImageSharp	https://github.com/SixLabors/ImageSharp/tree/5a7c1f4b2f2f96ccdc38a99d5130b3326d3958fb	Biblioteka za 2D grafiku	/	/	11
Ukupno			231	220	209

5.2 Anotaciona šema i smernice

Anotaciona šema korišćena prilikom validacije rešenja predstavljena je na Slici 5.1.



Slika 5.1 Anotaciona šema korišćena tokom validacije rešenja

Za svaki od pet mirisa koji su anotirani su definisane zasebne heuristike, dok su intenziteti za svaki miris isti. Za svaku heuristiku su navedena svojstva koda koja su anotatori analizirali kako bi odredili njenu primenjivost, a zatim su date smernice koje navode koji intenzitet dodeliti na osnovu primenjenih heuristika. Korišćene šeme i smernice su dostupne javnosti[26].

[26] Anotaciona šema i smernice za prvu fazu validacije se mogu pronaći na https://github.com/Clean-CaDET/CSharp_dataset_supplementary_material, dok su šema i smernice korišćene u drugoj fazi dostupne na https://github.com/Clean-CaDET/CSharp_dataset_prescriptive_procedure.

5.2.1 Heuristike za *dugačku metodu*

Tokom validacije rešenja, anotatori su anotirali miris *dugačka metoda* analizirajući primenjivost tri heuristike: *metoda je dugačka*, *metoda je kompleksna* i *metoda ima više odgovornosti*. U Tabeli 5.3 je istaknuta literatura koja ukazuje na relevantnost svake heuristike prilikom anotiranja *dugačke metode*. Pored toga, navedena su svojstva koda koja su anotatori analizirali kako bi odredili primenjivost svake heuristike.

Tabela 5.3 Heuristike korišćene prilikom anotacije *dugačke metode*.

Heuristika	Literatura	Validacija
Metoda je dugačka	Dužina metode je jednostavan pokazatelj potrebe za refaktorisanjem koda. Iskusni stručnjaci u industriji zastupaju mišljenje da metode treba da budu kratke [11][111], a istraživanja o percepciji mirisa od strane programera pokazuju da oni koriste dužinu metode prilikom određivanja prisustva mirisa *dugačka metoda* [21]. Takođe, autori pregleda literature [14] su pokazali da sva postojeća pravila za prepoznavanje *dugačke metode* koriste dužinu, tj., vrednost strukturne metrike koja predstavlja broj linija koda (engl. *Lines Of Code* - LOC).	Pri razmatranju ove heuristike, anotatori su analizirali dva svojstva koda: strukturnu metriku LOC i izvorni kod. Metrika LOC nije uvek dovoljna za precizno određivanje primenjivosti heuristike, jer je teško odrediti prag koji se posmatra za metriku [12]. Zatim, dužina može prevazilaziti prag, a da te linije koda predstavljaju niz jednostavnih operacija. Stoga, analiza izvornog koda može pomoći anotatorima u proceni da li svrha metode opravdava njenu dužinu. Ako su anotatori morali više puta da prelistavaju kod kako bi razumeli upotrebu podataka iz jednog dela metode u drugom, beležili su da je ova heuristika primenjiva. Sledeći pokazatelj su delovi koda koji se ponavljaju u više grana koda, što ukazuje na miris *ponovljeni kod*, koji se često javlja uz miris *dugačka metoda* [11][108]. Nepotrebni delovi koda (suvišne validacije, kod koji se nikad ne izvršava, podaci koji se ne koriste) takođe ukazuju na primenjivost ove heuristike.
Metoda je kompleksna	Uslovni blokovi i petlje koji povećavaju kompleksnost koda mogu ukazati na miris *dugačka metoda* [11]. Istraživanja pokazuju da programeri analiziraju kompleksnost koda kroz grananja i petlje prilikom prepoznavanja *dugačke metode* [21], a postojeća pravila za prepoznavanje *dugačke metode* takođe koriste strukturnu metriku za kompleksnost koda (engl. *Cyclomatic Complexity* - CYCLO) [14]. Posle LOC metrike, kompleksnost koda je druga najčešće korišćena metrika u pravilima za *dugačku metodu*.	Prilikom razmatranja kompleksnosti, anotatori su analizirali dva svojstva koda: strukturnu metriku CYCLO i izvorni kod. Vrednost CYCLO metrike predstavlja broj grananja, petlji, uslovnih izraza, logičkih operatora i drugih iskaza koji utiču na kompleksnost koda. Međutim, izazovno je definisati prag, a CYCLO metrika može ukazati na veći broj grananja koja imaju veoma jednostavne uslove, te ne otežavaju razumevanje i održavanje metode. Mentalni napor anotatora potreban za razumevanje metode je koristan pokazatelj njene kompleksnosti [112], te su se anotatori primarno oslanjali na izvorni kod prilikom određivanja primenjivosti ove heuristike. Anotatori nisu označavali ovu heuristiku primenjivom ako su analizom izvornog koda uočili velik broj grananja sa jednostavnim uslovima. Sa druge strane, ugnježdene petlje, grananja sa višestrukim i komplikovanim uslovima, dugačke pojedinačne linije koda sa mnoštvom ulančanih izraza, aritmetičkih operacija, logičkih i ternarnih operatora su ukazivali na primenjivost ove heuristike. Ulančani izrazi ukazuju na miris *lanci poruka* koji se često javlja uz *dugačke metode* [108]. Mentalni napor anotatora je izražen i ukoliko postoje delovi koda čija svrha nije sasvim jasna i ukoliko se u kodu koriste „magični brojevi“ za koje ne postoji jasno objašnjenje [111].
Metoda ima više odgovornosti	Stručnjaci u industriji ističu važnost principa jedinstvene odgovornosti (SRP) koji definiše da svaka komponenta u sistemu (npr. metoda ili klasa) treba da ima jednu odgovornost [113]. Empirijska istraživanja pokazuju da programeri analiziraju da li metoda ima više odgovornosti prilikom anotiranja *dugačke metode* [21]. Određivanje odgovornosti metode je podložno subjektivnoj interpretaciji, pošto ne postoji precizna definicija odgovornosti. Deo metode koji se bavi rukovanjem izuzecima se u jednom slučaju može posmatrati kao zasebna odgovornost, dok u drugom može predstavljati neizostavan deo metode. Treba uzeti u obzir koliko se deo koda semantički uklapa sa ostatkom koda i kolika je njegova kompleksnost. Indikatori odgovornosti mogu biti komentari i novi redovi koji dele metodu na segmente [11][114].	Tokom validacije rešenja, anotatori su analizirali izvorni kod i označavali heuristiku kao primenjivu kada pronađu delove metode koji semantički ne pripadaju klasi u kojoj se metoda nalazi. Umesto toga, delovi metode implementiraju logiku koja bi trebalo da se nalazi u drugoj klasi. Ovaj primer ukazuje na miris *zavist među odlikama*, koji se često javlja uz miris *dugačka metoda* [53][108]. Zatim, anotatori su tražili delove metode koji se ponavljaju uz minimalne razlike. Ovakvi delovi koda ukazuju na miris *ponovljeni kod* koji često koegzistira sa *dugačkom metodom* [11][108]. Sledeći pokazatelj primenjivosti heuristike su bili različiti nivoi apstrakcije [111], gde su izrazi niskog nivoa bili ugnježdeni među pozivima metoda višeg nivoa. Različiti nivoi apstrakcije ukazuju na pogrešno raspoređenu logiku ili nedostatak metoda višeg nivoa. Za kraj, u nekim metodama nije bilo očiglednih pokazatelja postojanja više odgovornosti. U tim slučajevima, anotatori su morali dublje da analiziraju semantiku metode i da izdvoje skrivene odgovornosti koje ona možda ima. Ovako teške primere je najčešće otkrivao anotator sa najviše iskustva.

5.2.2 Heuristike za *veliku klasu*

Tokom validacije rešenja, anotatori su anotirali miris *velika klasa* analizirajući primenjivost tri heuristike: *klasa je dugačka*, *klasa je kompleksna* i *klasa ima više odgovornosti*. U Tabeli 5.4 je za svaku heuristiku istaknuta literatura koja ukazuje na njenu relevantnost za anotiranje mirisa *velika klasa*. Navedena su i svojstva koda koja su anotatori analizirali kako bi odredili primenjivost heuristika.

Tabela 5.4 Heuristike korišćene prilikom anotacije *velike klase*.

Heuristika	Literatura	Validacija
Klasa je dugačka	*Velike klase* često imaju velik broj polja i metoda, kao i velik broj linija koda [11], a empirijska istraživanja pokazuju da programeri koriste dužinu klase prilikom određivanja prisustva mirisa *velika klasa* [21]. Pored toga, pregled literature [14] pokazuje da postojeća pravila za prepoznavanje *velike klase* često koriste strukturne metrike za broj metoda (engl. *Number Of Methods* - NOM), broj polja (engl. *Number Of Fields* – NOF ili *Number Of Attributes* - NOA) i broj linija koda (LOC).	Anotatori su se složili da je dužina klase značajna heuristika koja ukazuje na prisustvo mirisa *velika klasa*. Kako bi odredili da li je primenjiva u posmatranom isečku koda, anotatori su analizirali strukturne metrike NOM, NOF i LOC kako bi stekli generalni utisak o veličini klase koju analiziraju. Nisu se oslanjali samo na strukturne metrike, jer njihove vrednosti, pogotovo LOC, ne moraju ukazivati na kod koji je teško razumeti i održavati. Na primer, postoje klase koje imaju preko 300 linija koda, ali stil kojim su pisane podrazumeva prekomernu upotrebu praznih redova, razdvajanje konstrukcije objekata na više linija koda, pozivanje metoda u više linija koda i kratke lance poziva metoda. Pošto broj linija koda može biti obmanjujući, anotatori su analizirali i izvorni kod.
Klasa je kompleksna	Softverski inženjeri smatraju klasu kompleksnom kada imaju poteškoća u kreiranju mentalnog modela kako klasa funkcioniše [59]. Kompleksna klasa sadrži kompleksne metode [38], što pokazuju i pravila za prepoznavanje *velike klase* [14]. Pored broja metoda (NOM), druga najčešće korišćena metrika u pravilima se tiče kompleksnosti klase (engl. *Weighted Method Count* - WMC). Metrika WMC predstavlja zbir kompleksnosti svih metoda u klasi, a kompleksnost pojedinačne metode je izražena metrikom CYCLO.	Pored strukturne metrike WMC, anotatori su tokom validacije rešenja analizirali izvorni kod koji omogućava identifikovanje delova klase koji su kompleksni. Anotatori su označavali heuristiku *klasa je kompleksna* ako klasa sadrži nekoliko kompleksnih metoda. Ako klasa sadrži jednu kompleksnu metodu, ali je ostatak koda trivijalan (nekoliko polja i jednostavnih metoda), anotatori nisu označavali klasu kompleksnom. Sa druge strane, klasa je kompleksna ako pored jedne kompleksne metode postoji veći broj polja i netrivijalna unutrašnja klasa. Unutrašnje klase doprinose kompleksnosti klase, a pogotovo ako ih ima više. Zatim, nazivi koji se pojavljuju u kodu mogu uticati na složenost koda i razumevanje svrhe klase [11]. Misteriozni nazivi otežavaju razumevanje, dok delovi koda koji imaju jasne nazive smanjuju mentalni napor anotatora.
Klasa ima više odgovornosti	Princip jedinstvene odgovornosti (SRP) nalaže da klasa treba da ima jednu odgovornost. Prema tome, klasa treba da se menja samo iz jednog razloga ili u okviru jedne kategorije zahteva [113]. Empirijska istraživanja pokazuju da programeri prilikom anotiranja *velike klase* razmatraju da li je SRP princip ispoštovan [21]. Ako se odgovornost klase može opisati u jednoj rečenici bez korišćenja veznika, može se reći da je odgovornost klase jasna i jedinstvena [21][59][111]. Odgovornost koju klasa ima podrazumeva da klasa poznaje detalje logike neophodne za tu odgovornost [113]. Prema tome, koordinatorske klase ne moraju nužno kršiti SRP princip, ukoliko ne zalaze u logiku drugih klasa, već samo delegiraju zadatke drugim klasama [38][59].	Anotatori su tokom validacije rešenja pokušavali da u izvornom kodu prepoznaju podskupove polja i metoda koji semantički mogu da se grupišu i izdvoje iz posmatrane klase u posebnu klasu. Zatim, anotatori su se oslanjali na komentare i prazne redove koji potencijalno mogu ukazati na različite odgovornosti u klasi. Slično tome, prefiksi u nazivima polja ili metoda mogu biti značajni za otkrivanje skrivenih odgovornosti klase. Za kraj, anotatori su analizirali dugačke i kompleksne metode kako bi utvrdili odgovornosti koje one implementiraju.

5.2.3 Heuristike za *klasu podataka*

Tokom validacije rešenja, anotatori su anotirali miris *klasa podataka* analizirajući primenjivost dve heuristike: *metode ne sadrže logiku* i *druge klase koriste posmatranu klasu*. U Tabeli 5.5 je istaknuta literatura koja ukazuje na relevantnost ovih heuristika za anotiranje mirisa *klasa podataka*. Pored toga, navedena su svojstva koda koja su anotatori analizirali kako bi odredili primenjivost heuristika.

Tabela 5.5 Heuristike korišćene prilikom anotacije *klase podataka*.

Heuristika	Literatura	Validacija
Metode ne sadrže logiku	Empirijska istraživanja pokazuju da programeri prilikom prepoznavanja *klase podataka* analiziraju metode klase u potrazi za logikom [21]. Ukoliko klasa većinski poseduje javna polja, *get* i *set* metode, programeri je smatraju *klasom podataka* [21]. Slično tome, pravila za prepoznavanje ovog mirisa koda najčešće koriste strukturne metrike koje se odnose na broj *get* i *set* metoda (engl. *Number Of Accessor Methods* - NOAM) i broj javnih polja (engl. *Number Of Public Attributes* – NOPA) [14], kao i WOC (engl. *Weight Of Class*) koja predstavlja odnos metoda koje implementiraju logiku i članova koji ne implementiraju logiku.	Tokom validacije rešenja, anotatori su analizirali nabrojane strukturne metrike i izvorni kod kako bi odredili primenjivost heuristike *metode ne sadrže logiku*. Pošto strukturne metrike pružaju samo numeričke vrednosti, izvorni kod omogućava dublju analizu logike koja je implementirana u klasi. Anotatori su u potrazi za logikom tražili grananja, logičke izraze i petlje u okviru funkcionalnih metoda. U funkcionalne metode ne spadaju *get*, *set*, *equals*, *hash*, *toString* i apstraktne metode. Anotatori su heuristiku *metode ne sadrže logiku* označavali kao primenjivu ukoliko većina metoda ne sadrži logiku.
Druge klase koriste posmatranu klasu	*Klase podataka* ne sadrže logiku za upravljanje svojim podacima, te predstavljaju kontejnere podataka kojima manipulišu druge klase u sistemu [21][45][53][65]. Kako bi se utvrdila uloga *klase podataka* u sistemu, potrebno je analizirati i klijentske klase koje pristupaju njenim podacima. Prema tome, analiza *klase podataka* kroz strukturne metrike potencijalno nije dovoljno precizna, jer je za ovaj miris koda potrebno razumeti i ponašanje, a ne samo strukturne karakteristike. Sa druge strane, analiza izvornog koda svih povezanih isečaka (potencijalne *klase podataka* i klijentskih klasa) može biti izuzetno vremenski zahtevna.	Tokom validacije rešenja, anotatori su analizirali svojstvo koda koje predstavlja veze među komponentama sistema. Ovo svojstvo omogućava dublju analizu od strukturnih metrika, ali nije vremenski zahtevno koliko analiza izvornog koda, jer izdvaja relevantne informacije o vezama među klasama, a uklanja nepotrebne detalje implementacije. Pomoću ovog svojstva, anotatori su proučavali koje klijentske klase koriste posmatranu klasu i na koji način (da li koriste javna polja, *get* i *set* metode posmatrane klase). Metode koje pripadaju klijentskim klasama i upravljaju podacima posmatrane klase ukazuju na miris *zavist među odlikama* koji se često javlja uz *klasu podataka* [53][108].

5.2.4 Heuristike za *zavist među odlikama*

Tokom validacije rešenja, anotatori su anotirali miris *zavist među odlikama* analizirajući primenjivost četiri heuristike: *metoda sadrži lance poziva metoda*, *metoda koristi tuđe podatke*, *metoda koristi podatke iz samo par drugih klasa* i *metoda semantički ne pripada klasi*. Za svaku heuristiku je u Tabeli 5.6 istaknuta literatura koja ukazuje na relevantnost za anotiranje mirisa *zavist među odlikama*, kao i svojstva koda koja su anotatori analizirali kako bi odredili primenjivost heuristika.

Tabela 5.6 Heuristike korišćene prilikom anotacije *zavisti među odlikama*.

Heuristika	Literatura	Validacija
Metoda sadrži lance poziva metoda	Metode koje su zahvaćene mirisom *zavist među odlikama* često pozivaju metode drugih klasa [14][57]. Lanac poziva metoda se javlja kada se poziva više metoda jedna za drugom u istom izrazu. Svaka metoda u lancu vraća objekat nad kojim se poziva sledeća metoda. Ovakvi izrazi ukazuju na to da metoda previše zavisi od podataka i funkcionalnosti drugih klasa.	Tokom validacije rešenja, anotatori su analizirali izvorni kod u potrazi za lancima poziva metoda. Najmanji lanac poziva treba da ima dva uzastopna poziva metoda. Najočigledniji lanci se prepoznaju po mnoštvu tačaka u izrazu, gde se metode pozivaju direktno nad objektom dobijenim iz prethodnog poziva metode. Međutim, postoje i skriveni lanci poziva metoda, gde je jedan izraz razložen na nekoliko linija koda. Na primer, objekat koji je vratila jedna metoda se sačuva u promenljivoj, a tek u narednom redu se nad tom promenljivom poziva sledeća metoda.
Metoda koristi tuđe podatke	Empirijska istraživanja pokazuju da programeri prilikom anotiranja *zavisti među odlikama* analiziraju da li metoda pristupa i manipuliše podacima drugih klasa putem javnih polja ili *get* i *set* metoda [21]. Pored toga, programeri istražuju i da li metoda više koristi tuđe podatke u odnosu na one koji se nalaze u njenoj klasi [21]. Slično tome, pravila za prepoznavanje *zavisti među odlikama* najčešće koriste strukturne metrike kojima se broje pristupi spoljašnjim metodama i podacima (engl. *External Method Call* – xMC, *Access of Import Data* – AID, *Access To Foreign Data* – ATFD) i lokalnim metodama i podacima (engl. *Internal Method Call* – IMC, *Access of Local Data* – ALD) [14]. Još jedna često korišćena metrika je LCOM (engl. *Lack of Cohesion between Methods*) koja može ukazati na nisku kohezivnost klase, a u kombinaciji sa CBO (engl. *Coupling Between Objects*) i na veliku zavisnost od drugih klasa u sistemu. Istraživanje o percepciji mirisa pokazuje i da programeri prilikom analize *zavisti među odlikama* istražuju da li je prekršen Demetrin zakon (engl. *Law of Demeter*) [21]. Demetrin zakon kaže da metode mogu pristupati podacima i metodama klase u kojoj se nalaze, kao i podacima i metodama iz parametara i objekata kreiranim u metodi.	Prilikom validacije rešenja, anotatori su analizirali veze među komponentama sistema kako bi proučili čijim podacima metoda pristupa i u kojoj meri. Pored toga, anotatori bi analizirali izvorni kod posmatrane metode kako bi detaljnije ispitali na koji način metoda pristupa tuđim podacima. Anotatori nisu uzimali u obzir pristupanje podacima putem parametara koje je metoda dobila ili objekata koji su kreirani u metodi, jer ovakav način pristupa podacima ne narušava Demetrin zakon. Ukoliko metoda više koristi tuđe podatke nego lokalne, anotatori bi označavali heuristiku *metoda koristi tuđe podatke* kao primenjivu.
Metoda koristi podatke iz samo par drugih klasa	Pored toga što metoda pristupa tuđim podacima, bitno je odrediti i kojoj klasi „zavidi“ metoda koja pati od *zavisti među odlikama*. Na osnovu te informacije je moguće refaktorisati kod i spojiti metodu sa podacima kojima upravlja. Ukoliko metoda koristi podatke iz mnogo drugih klasa, onda nije moguće tačno definisati kojoj od njih „zavidi“. U pravilima za prepoznavanje *zavisti među odlikama* se koriste strukturne metrike za broj klasa čije podatke koristi posmatrana metoda (engl. *Number of Import Classes* – NIC, *Foreigh Data Providers* – FDP) [14]. Ova pravila proveravaju da li je vrednost navedenih strukturnih metrika relativno mala (manja od 3).	Tokom validacije rešenja, anotatori su analizirali svojstvo veze među komponentama kako bi odredili klasu kojoj posmatrana metoda „zavidi“. Ukoliko metoda koristi podatke iz mnogih klasa, anotatori nisu označavali heuristiku *metoda koristi podatke iz samo par drugih klasa* primenjivom. Pored toga, anotatori su analizirali izvorni kod klasa kojima potencijalno metoda „zavidi“, kako bi potvrdili da se podaci i metoda semantički uklapaju.
Metoda semantički ne pripada klasi	Pored broja pristupa tuđim podacima i određivanja klasa kojima metoda „zavidi“, važno je razmotriti i da li metoda semantički pripada klasi u kojoj se nalazi. Moguće je da metoda nema mnogo zavisnosti sa drugim klasama ili da jednako zavidi podacima nekoliko klasa, ali da se na osnovu semantike klasa može zaključiti u koju klasu ju je najprikladnije smestiti [114]. Metoda treba da se nalazi u klasi čije izmene bi uticale i na nju [57].	Analizom veza među komponentama i izvornog koda, anotatori su tokom validacije rešenja označavali heuristiku *metoda semantički ne pripada klasi* kada su uspevali da odrede klasu u koju metodu treba smestiti. Osim toga, moguće je da metoda ne pripada nijednoj postojećoj klasi, već je potrebno kreirati novu klasu koja će enkapsulirati određene podatke i funkcionalnosti, među kojima bi bila i posmatrana metoda.

5.2.5 Heuristike za *odbačeno nasledstvo*

Tokom validacije rešenja, anotatori su anotirali miris *odbačeno nasledstvo* analizirajući primenjivost tri heuristike: *klasa retko koristi nasleđene članove*, *mnogi nasleđeni članovi su redefinisani* i *hijerarhija nasleđivanja je nepotrebna*. U Tabeli 5.7 je za svaku heuristiku navedena literatura koja ukazuje na relevantnost za anotiranje mirisa *odbačeno nasledstvo*, kao i svojstva koda koja su anotatori analizirali kako bi odredili primenjivost heuristika.

Tabela 5.7 Heuristike korišćene prilikom anotacije *odbačenog nasledstva*.

Heuristika	Literatura	Validacija
Klasa retko koristi nasleđene članove	Empirijska istraživanja pokazuju da programeri analiziraju da li klasa koristi nasleđene članove (polja i metode) prilikom anotiranja *odbačenog nasledstva* [21]. Slično tome, pravila za prepoznavanje *odbačenog nasledstva* koriste strukturnu metriku BUR (engl. *Base-class Usage Ratio*). BUR metrika računa odnos upotrebljenih nasleđenih članova i ukupnog broja nasleđenih članova.	Pored strukturne metrike BUR, anotatori su tokom validacije rešenja analizirali veze među komponentama i izvorni kod kako bi bolje razumeli semantiku iza hijerarhije u kojoj se nalazi posmatrana klasa. Pored informacije da li posmatrana klasa koristi nasleđene članove, anotatori su analizirali i da li klijentske klase posmatrane klase koriste članove koje je ona nasledila. Pored istraživanja da li klasa koristi članove roditeljske klase, anotatori su razmotrili i da li koristi članove drugih klasa u hijerarhiji (iznad roditeljske klase).
Mnogi nasleđeni članovi su redefinisani	*Odbačeno nasledstvo* se može prepoznati kada klasa naslednica redefiniše mnoge nasleđene članove [24][58]. Pravila za prepoznavanje *odbačenog nasledstva* takođe koriste strukturnu metriku koja meri broj redefinisanih metoda u odnosu na ukupan broj metoda u klasi (engl. *Base-class Overriding Ratio* – BOvR) [14].	Tokom validacije rešenja, anotatori su analizirali izvorni kod kako bi utvrdili koliko nasleđenih metoda je redefinisano u posmatranoj klasi. Anotatori nisu označavali heuristiku *mnogi nasleđeni članovi su redefinisani* ukoliko su redefinisane metode za koje se to podrazumeva (metode interfejsa ili apstraktne metode).
Hijerarhija nasleđivanja je nepotrebna	Odnos roditeljske klase i njenih potomaka treba da bude značajniji od saradnje dve klase koje nisu u hijerarhiji nasleđivanja [53]. Ova saradnja se temelji na skupu članova (polja i metoda) koje roditeljska klasa priprema specijalno za upotrebu od strane svojih naslednika. Ako klasa naslednica ne koristi ili redefiniše većinu nasleđenih članova, hijerarhija je očigledno nepotrebna. Međutim, to nije uvek tako očigledno.	Tokom validacije rešenja, anotatori su analizirali izvorni kod i veze među komponentama, kako bi utvrdili da li hijerarhija nasleđivanja ima smisla i da li klase, na osnovu semantike, treba da budu deo nje. Čak i ako klase u nekoj meri koriste nasleđene članove, anotatori su analizom izvornog koda i veza među komponentama mogli proceniti da li bi se dizajn sistema mogao unaprediti izbacivanjem određenih klasa iz hijerarhije, premeštanjem klasa u druge hijerarhije ili potpunim uklanjanjem postojeće hijerarhije.

5.2.6 Vrednosti intenziteta mirisa koda

U sistematskom pregledu literature [17] istaknut je značaj intenziteta za određivanje prioriteta uklanjanja mirisa koda. Iako mnoga istraživanja vrše automatsku dodelu intenziteta, njihovi rezultati ukazuju na neslaganje oko faktora koji se koriste za prioritizaciju mirisa [69]. Nesuglasice bi se mogle umanjiti empirijskim istraživanjima koja bi pokazala koje faktore programeri koriste prilikom prioritizacije mirisa [69].

Empirijska istraživanja koja ispituju problem prioritizacije mirisa su retka. U dva istraživanja su programeri određivali kritičnost mirisa na skali od 1 do 5 [23][96]. Skalu sa četiri nivoa su koristili anotatori u istraživanju [27]. Međutim, u pomenutim istraživanjima nisu navedeni faktori koje treba razmotriti prilikom određivanja intenziteta. Po uzoru na istraživanja [13] i [88] koja su obezbedila opis korišćenih intenziteta, u ovom istraživanju je definisana skala intenziteta koja se može prilagoditi potrebama interesnih grupa. Skala intenziteta sadrži četiri vrednosti, a njihove definicije su date u Tabeli 5.8.

Tabela 5.8 Opis vrednosti intenziteta korišćenih tokom validacije rešenja, zasnovanih na skali intenziteta iz prethodnih istraživanja [13][88]

Vrednost intenziteta	Definicija u [13] i [88]	Definicija u ovom istraživanju
0	Nema mirisa koda: klasa/funkcija nije zahvaćena mirisom	Miris ne postoji ili ima zanemarljiv uticaj. Isečak koda (klasa ili funkcija) ne mora biti savršen i može postojati prostor za sitna unapređenja, ali nije neophodno refaktorisanje.
1	Bezopasan miris koda: klasa/funkcija je delimično zahvaćena mirisom	Prisutan je miris koji blago narušava održivost koda. Jedna ili dve akcije refaktorisanja su dovoljne za uklanjanje mirisa. U pogledu prioritizacije uklanjanja, ovakav miris ne zahteva hitno refaktorisanje, ali bi ga trebalo ukloniti blagovremeno. Veći broj isečaka zahvaćenih mirisima ovog intenziteta mogu povećati napor neophodan za razumevanje koda.
2	Miris koda: klasa/funkcija pokazuje jasne znake prisustva mirisa	Miris značajno narušava održivost koda. Potreban je niz akcija refaktorisanja kako bi se uklonio. Uklanjanje mirisa ovog intenziteta treba prioritizovati, pogotovo ako je isečak koda deo softvera koji se aktivno razvija, pošto negativno utiče na svakodnevni razvoj i održavanje softvera.
3	Veoma štetan miris koda: miris je prisutan i značajno otežava održivost	Miris kritično narušava održivost koda. Uklanjanje ovakvog mirisa je neophodno i zahteva posvećen rad na ponovnom dizajniranju softvera i niz ozbiljnih promena u kodu. Takođe, uklanjanje ovog mirisa je od najvećeg prioriteta.

Skala intenziteta opisana u Tabeli 5.8 je korišćena prilikom anotiranja pet mirisa koda tokom validacije rešenja. U Tabeli 5.9 su date smernice koje su anotatori pratili tokom dodeljivanja intenziteta za svaki miris koda. Važno je istaći da su opisane smernice služile kao okvirna pravila za dodeljivanje intenziteta. Na primer, za miris *dugačka metoda* je navedeno da na intenzitet 1 ukazuje metoda koja ima više od 10 linija koda. Pošto je ovo okvirno pravilo, anotator može dodeliti intenzitet 1 čak i ako metoda ima manje linija koda, ukoliko druge heuristike ukazuju na to da metoda narušava održivost koda, te da bi je blagovremeno trebalo refaktorisati.

Tabela 5.9 Smernice za dodeljivanje intenziteta

Intenzitet	Dugačka metoda	Velika klasa	Klasa podataka	Zavist među odlikama	Odbačeno nasledstvo
0	Nijedna heuristika nije primenjiva ili su neke primenjive. Vrednosti u heuristikama ne prevazilaze vrednosti date za intenzitet 1.				
1	Neke ili sve heuristike su primenjive: - Metoda ima 10+ linija koda - Metoda ima skromnu složenost (npr. ima uslovne blokove, petlje,...) - Metoda ima 2-3 odgovornosti koje se mogu izdvojiti u zasebne metode	Neke ili sve heuristike su primenjive: - Klasa ima dve odgovornosti - Klasa ima skromnu složenost (1-2 složene funkcije) - Klasa ima oko 100 linijja koda	Neke ili sve heuristike su primenjive: - Klasa ima oko 3 javna atributa - Klasa ima oko 3 *get* i *set* metode - Par drugih klasa (oko 3) koristi posmatranu klasu	Neke ili sve heuristike su primenjive: - Metoda ima oko 10 kratkih lanaca poziva metoda (2-3 povezane metode) - Metoda ima oko 2 srednja lanca poziva metoda (4-5 povezanih metoda) - Metoda nema dugačke lance poziva metoda (6-7 povezanih metoda) - Metoda koristi oko 5 spoljašnjih podataka (metode, *get* ili *set* metode, polja)	Neke ili sve heuristike su primenjive: - Klasa redefiniše 30-50% nasleđenih metoda - Klasa koristi 30-50% nasleđenih metoda i podataka
2	Neke ili sve heuristike su primenjive: - Metoda ima nekoliko desetina linija koda - Metoda je složena i zahteva više vremena i truda za razumevanje (npr. ima ugnježdene petlje) - Metoda ima 4-5 odgovornosti koje se mogu izdvojiti u zasebne metode	Neke ili sve heuristike su primenjive: - Klasa ima 3-4 odgovornosti - Klasa ima 3+ složene funkcije - Klasa ima nekoliko stotina linija koda	Neke ili sve heuristike su primenjive: - Klasa ima oko 5 javnih atributa - Klasa ima oko 7 *get* i *set* metoda - Nekoliko drugih klasa (oko 6) koristi posmatranu klasu	Neke ili sve heuristike su primenjive: - Metoda ima oko 15 kratkih lanaca poziva metoda (2-3 povezane metode) - Metoda ima oko 10 srednjih lanaca poziva metoda (4-5 povezanih metoda) - Metoda ima oko 2 dugačka lanca poziva metoda (6-7 povezanih metoda) - Metoda koristi oko 15 spoljašnjih podataka (metode, *get* ili *set* metode, polja)	Neke ili sve heuristike su primenjive: - Klasa redefiniše 50-70% nasleđenih metoda - Klasa koristi 50-70% nasleđenih metoda i podataka
3	Neke ili sve heuristike su primenjive. Vrednosti u heuristikama prevazilaze vrednosti date za intenzitet 2.				

5.3 Skup podataka

U Tabeli 5.10 su predstavljene osnovne karakteristike skupa podataka nastalog kao rezultat validacije rešenja. Skup sadrži ukupno 4154 anotirana isečka koda, a u tabeli se može videti broj anotiranih isečaka koda po mirisu.

Za svaki miris u skupu podataka je objavljena datoteka sa strukturnim metrikama, pojedinačnim labelama i konačnom labelom za svaki isečak koda. Pored datoteka sa labelama i strukturnim metrikama, skup podataka sadrži i datoteke sa heuristikama, kako bi se mogli analizirati razlozi zbog kojih su anotatori dodelili pojedinačne labele. Za svaki isečak je navedeno koje heuristike je svaki pojedinačni anotator naveo kao primenjive.

Strukturne metrike su izračunate pomoću Clean CaDET platforme, a u Tabeli 5.10 se može videti broj strukturnih metrika po mirisu. U prvoj fazi validacije rešenja, platforma je omogućila izračunavanje 18 strukturnih metrika za metode i 25 za klase. Platforma je zatim unapređena, te je u drugoj fazi validacije rešenja vršila izračunavanje 19 strukturnih metrika za metode i 31 za klase. U Tabeli 5.11 se nalaze linkovi koji vode do javno dostupnog skupa podataka, platforme upotrebljene za izračunavanje strukturnih metrika, definicija metrika i njihove implementacije.

Tabela 5.10 Osnovne karakteristike skupa podataka

	Prva faza validacije		**Druga faza validacije**		
	Dugačka metoda	**Velika klasa**	**Klasa podataka**	**Zavist među odlikama**	**Odbačeno nasledstvo**
Broj anotiranih isečaka koda	2574	920	231	220	209
Broj strukturnih metrika	18	25	31	19	31

Tabela 5.11 Korisni resursi vezani za skup podataka

Resurs	**Link**
Skup podataka iz prve faze validacije	https://zenodo.org/records/6520056#.Y2O1wnbMKUk
Skup podataka iz druge faze validacije	https://doi.org/10.5281/zenodo.10475432
Clean CaDET platforma korišćena za računanje strukturnih metrika	https://github.com/Clean-CaDET/platform#readme
Spisak strukturnih metrika	https://github.com/Clean-CaDET/platform/blob/master/CodeModel/CaDETModel/CodeItems/CaDETMetrics.cs
Implementacija strukturnih metrika za klase	https://github.com/Clean-CaDET/platform/blob/master/CodeModel/CodeParsers/CSharp/CaDETClassMetricCalculator.cs
Implementacija strukturnih metrika za metode	https://github.com/Clean-CaDET/platform/blob/master/CodeModel/CodeParsers/CSharp/CaDETMemberMetricCalculator.cs

Nad anotiranim skupom podataka se može izvršiti analiza performansi anotatora. Skup podataka sadrži konačne labele, dobijene na osnovu pojedinačnih labela koje su pojedinačni anotatori dodeljivali za svaki isečak koda. Konačna labela sadrži intenzitet mirisa koji je većinski bio zastupljen u pojedinačnim labelama. Na osnovu pojedinačnih i konačnih labela je moguće analizirati performanse pojedinačnih anotatora na dva načina:

- posmatrajući konačnu labelu kao referentnu istinu
- posmatrajući labelu najiskusnijeg anotatora kao referentnu istinu

U Tabeli 5.12 su izlistane F-mere koje su anotatori postigli prilikom anotiranja mirisa koda. Analiza performansi anotatora je izuzetno važna za izgradnju ML modela, jer omogućava postavljanje očekivanih rezultata kojima se teži [119]. Performanse ML modela se mogu unaprediti ukoliko postoje primeri na kojima model pravi greške, a čovek ne, i ukoliko su poznati razlozi zašto su te greške nastale [119]. Razumevanje razloga je moguće ukoliko postoje detaljne anotacione smernice koje daju uvid u rezonovanje anotatora koje se krije iza dodeljenih anotacija. Iz ovog razloga, skup podataka koji je rezultat predloženog rešenja sadrži heuristike koje su anotatori označili kao prisutne i na osnovu kojih su dodelili anotacije. Primenjene heuristike doprinose boljem razumevanju odluka anotatora.

Tabela 5.12 Poređenje performansi (F-mere) anotatora.

Referentna istina	**Miris koda**	**Anotator 1**	**Anotator 2**	**Anotator 3**
Konačna labela	Dugačka metoda	0.96	0.9	0.85
	Velika klasa	0.95	0.91	0.88
	Klasa podataka	0.98	0.98	0.98
	Zavist među odlikama	0.93	0.96	0.94
	Odbačeno nasledstvo	0.98	0.98	0.83
Labela najiskusnijeg anotatora (Anotator 1)	Dugačka metoda	/	0.8	0.73
	Velika klasa	/	0.79	0.75
	Klasa podataka	/	0.95	0.79
	Zavist među odlikama	/	0.88	0.82
	Odbačeno nasledstvo	/	0.95	0.5

Analiza performansi anotatora je posebno važna za mirise kao što su *klasa podataka, zavist među odlikama* i *odbačeno nasledstvo*, za koje pravila za detekciju zasnovana na strukturnim metrikama imaju loše performanse (detaljnije opisano u Potpoglavlju 5.4.1). Pored toga, ove mirise koda je teže otkriti pomoću ML modela zasnovanih na strukturnim metrikama, te se moraju upotrebiti druge karakteristike ili modeli zasnovani na dubokom učenju kao što je *code embedding* koji mogu sami izvući relevantne karakteristike. Problem takvog pristupa je što zahteva velik skup podataka.

U Tabeli 5.13 su upoređene performanse (F-mere) ML modela obučenih na dva skupa podataka, MLCQ skupu [27] i skupu koji je nastao primenom rešenja za anotiranje predloženog u ovom istraživanju[27]. Izlistani su rezultati tri istraživanja koja su koristila MLCQ skup. Performanse ovih istraživanja su slične, iako koriste različite atribute i ML modele (navedene u tabeli). Nijedno od ovih istraživanja nije uzelo u obzir miris *odbačeno nasledstvo*.

Performanse ML modela pri detekciji *dugačke metode* i *velike klase* su više u istraživanjima koje koriste drugi skup podataka, nastao primenom rešenja predloženog u ovom istraživanju. Pošto ovaj skup ima konzistentnije anotacije od MLCQ skupa (detaljnije u Potpoglavlju 5.4.1), ovakve performanse su bile očekivane. Međutim, iako su anotacije za *klasu podataka* i *zavist među odlikama* konzistentnije od MLCQ skupa, performanse ML modela obučenih na ovom skupu podataka su slične performansama prijavljenim u istraživanjima koje koriste MLCQ skup. Loše performanse ML modela pri prepoznavanju *klase podataka i zavisti među odlikama* su analizirane u Potpoglavlju 6.3 koje obrađuje pretnje validnosti.

[27] Tabela 5.11 sadrži korisne resurse vezane za skup podataka.

Tabela 5.13 Poređenje performansi (F-mere) ML modela obučenih na dva različita skupa podataka (MLCQ [27] i skup koji je rezultat predloženog rešenja za anotiranje).

Skup podataka	Istraživanje koje je objavilo rezultate	Dugačka metoda	Velika klasa	Klasa podataka	Zavist među odlikama
MLCQ skup [27]	[122]	0.77 (pristup: *Random Forest*)	0.57 (pristup: *Random Forest*)	**0.63** (pristup: *Random Forest*)	**0.32** (pristup: *Random Forest*)
	[15]	0.75 (pristup: XGBoost + SMOTEENN, atributi: CuBERT source code embeddings)	0.53 (pristup: Bagging (SVM) + SMOTEENN, atributi: CuBERT source code embeddings)	/	/
	[104]	/	/	0.62 (pristup: *Random Forest*, atributi: strukturne metrike)	0.26 (pristup: *Random Forest*, atributi: codeT5_small)
Skup iz ovog istraživanja	[71]	**0.87** (pristup: Catboost, atributi: strukturne metrike)	**0.88** (pristup: *Random Forest*, atributi: strukturne metrike)	/	/
	Ovo istraživanje	/	/	0.17 (pristup: *Random Forest*, atributi: strukturne metrike)	0.25 (pristup: *Random Forest*, atributi: codeT5_small)

5.4 Validacija zahteva

Na osnovu sprovedenog eksperimenta je potrebno utvrditi da li rešenje ispunjava zahteve definisane u Tabeli 1.1. U narednim potpoglavljima je analizirana ispunjenost navedenih zahteva.

5.4.1 Ispunjenost zahteva 1

Prvi zahtev se odnosi na uključivanje obuke anotatora u anotacionu proceduru. Ovaj zahtev adresira fenomen nekonzistentnih anotacija koje narušavaju kvalitet skupa podataka i otežavaju obučavanje ML modela. Obuka anotatora treba da poveća slaganje anotatora i doprinese konzistentnosti anotacija.

Obuka anotatora je jedna od aktivnosti anotacione procedure (opisana u Potpoglavlju 4.2.3). Podržana je konceptualnim modelom koji definiše *anotacionu šemu* i *anotacione smernice* (opisane u Potpoglavlju 4.1). Anotaciona šema sinhronizuje razumevanje anotatora, a smernice upućuju anotatore kako da primene šemu tokom anotiranja. DSE alat omogućava definisanje i pregled anotacione šeme, pristup anotacionim smernicama, probno testiranje i identifikovanje konfliktnih anotacija koje treba prodiskutovati.

Za procenu konzistentnosti anotacija izračunato je slaganje među anotatorima (engl. *Inter-Annotator Agreement* - IAA). IAA se u NLP oblasti tipično koristi kao metrika za procenu koliko je precizno definisan zadatak anotacije. Visok IAA rezultat ukazuje na to da je anotaciona procedura pouzdana, jer anotatori generišu konzistentne anotacije [42]. U Tabeli 5.14 poredimo IAA rezultate sa MLCQ skupom podataka, čiji anotatori nisu prošli kroz obuku anotatora [27]. Rezultati u tabeli ukazuju na nizak nivo slaganja među anotatorima MLCQ skupa, dok visok IAA rezultat za skup podataka u ovom istraživanju ukazuje da je anotaciona procedura pouzdana, odnosno, ponovljiva, jer anotatori generišu konzistentne anotacije [42].

Tabela 5.14 Poređenje slaganja anotatora sa MLCQ skupom podataka [27]. Za poređenje se koristi *Krippendorff* alfa vrednost [115] u opsegu [-1, 1]. Landis i Koch [116] su kreirali kategorije koje služe za interpretaciju ovih vrednosti: *poor*, *slight*, *fair*, *moderate*, *substantial* i *almost perfect*.

Skup podataka	Dugačka metoda	Velika klasa	Klasa podataka	Zavist među odlikama	Odbačeno nasledstvo
Skup podataka iz ovog istraživanja	0.792 (*substantial agreement*)	0.841 (*almost perfect agreement*)	0.705 (*substantial agreement*)	0.644 (*substantial agreement*)	0.645 (*substantial agreement*)
MLCQ skup podataka	0.18 (*slight agreement*)	0.038 (*slight agreement*)	0.291 (*fair agreement*)	-0.005 (*poor agreement*)	/

Iako visok IAA rezultat pokazuje da je anotacioni zadatak precizno definisan [42], ne može se utvrditi da li ta definicija odgovara realnom problemu anotacije mirisa koda. Stoga se može proveriti da li su anotacije konzistentne u pogledu relevantnih svojstava koda. Kao automatski merljiva svojstva koda se mogu koristiti strukturne metrike koje su relevantne za dati miris koda. Iako se mirisi koda ne mogu precizno detektovati korišćenjem pravila zasnovanih na strukturnim metrikama, prisustvo mirisa bi trebalo da bude u korelaciji sa relevantnim metrikama [11]. Mäntylä i Lassenius [82] takođe ističu da bi anotacije anotatora trebalo da budu u korelaciji sa strukturnim metrikama.

Za utvrđivanje konzistentnosti anotacija se može koristiti statistički test multivarijantne analize varijanse sa ponovljenim merenjima (engl. *Multivariate Analysis of Variance* - MANOVA) [117] iz biblioteke *Statsmodels* [118]. MANOVA test je upotrebljen za mirise *dugačka metoda* i *velika klasa*, za koje ima smisla proveravati korelaciju anotacija i strukturnih metrika zato što postoje opšteprihvaćene strukturne metrike za njihovo prepoznavanje [14].

U Tabeli 5.15 su izlistane strukturne metrike korišćene u MANOVA testu, a izračunate pomoću *Clean CaDET* platforme[28]. Izabrane su metrike koje se koriste u pravilima [14] i ML modelima [12] za detekciju ovih mirisa. Mnoge od ovih strukturnih metrika su definisane u *CK metrics suite* [121] i široko prihvaćene u istraživačkoj zajednici koja se bavi detekcijom mirisa.

Tabela 5.15 Strukturne metrike korišćene u MANOVA testu.

Miris koda	Strukturne metrike
Dugačka metoda	CYCLO, CYCLO_SWITCH, MLOC, MELOC, NOP, NOLV, NOTC, MNOL, NOR, MNOC, MNOA, NONL, NOSL, NOMO, NOPE, NOLE, MMNB, NOUW
Velika klasa	CLOC, CELOC, LCOM, LCOM3, LCOM4, NMD, NAD, NMD_NAD, WMC, WMC_NO_CASE, ATFD, ATFD_10, TCC, CNOR, CNOL, CNOC, CNOA, NOPM, NOPF, DIT, DCC, CMNB, RFC, CBO, NIC

Postoje dva aspekta konzistentnosti koji se mogu razmotriti u pogledu strukturnih metrika, tj., da li:

1. Anotacije svakog pojedinačnog anotatora koreliraju sa relevantnim strukturnim metrikama (anotator treba da dodeli isti intenzitet isečcima koda koji su slični u pogledu vrednosti strukturnih metrika)
2. Anotacije različitih anotatora su međusobno konzistentne u pogledu relevantnih strukturnih metrika (različiti anotatori treba da dodele isti intenzitet isečcima koda koji su slični u pogledu vrednosti strukturnih metrika)

[28] Definicije strukturnih metrika dostupne na https://github.com/Clean-CaDET/platform/blob/c4acff95ec00ff6c25fa62dde4818c1f40e39d39/CodeModel/CaDETModel/CodeItems/CaDETMetrics.cs.

U prvom slučaju, za svakog anotatora, isečci koda se grupišu u kategorije koje odgovaraju intenzitetu mirisa koda koje je anotator dodelio. Intenzitet mirisa je nezavisna varijabla, dok su strukturne metrike zavisne varijable. Za *dugačku metodu* je uzeto u obzir 18 strukturnih metrika, a za *veliku klasu* 25[29]. MANOVA testom se utvrđuje da li postoji značajna razlika između kategorija, pri čemu se povoljnim ishodom smatra da razlika postoji. U Tabeli 5.16 su prikazani rezultati ovog testa za tri anotatora. Pošto je $p < 0.05$, zaključuje se da anotacije pojedinačnih anotatora koreliraju sa vrednostima strukturnih metrika, što je povoljan ishod testa.

Tabela 5.16 Rezultati MANOVA testa pokazuju da postoje značajne razlike između kategorija koje odgovaraju intenzitetima, pošto je $p < 0.05$. Prema tome, anotacije koreliraju sa vrednostima strukturnih metrika.

Miris koda	**Anotator**		
	1	**2**	**3**
Dugačka metoda	Wilks' lambda = 0.337 $F(17,1875) = 216.8$ $p = 0.000$	Wilks' lambda = 0.388 $F(17,2066) = 191.9$ $p = 0.000$	Wilks' lambda = 0.443 $F(17,2014) = 149.1$ $p = 0.000$
Velika klasa	Wilks' lambda = 0.341 $F(24,611) = 49.12$ $p = 0.000$	Wilks' lambda = 0.275 $F(24,707) = 77.7$ $p = 0.000$	Wilks' lambda = 0.338 $F(24,631) = 51.4$ $p = 0.000$

U drugom slučaju, za svaki intenzitet mirisa, isečci se grupišu po anotatoru koji je dodelio taj intenzitet. Anotator je nezavisna varijabla, dok su strukturne metrike zavisne varijable. Odabrane metrike su iste kao u prvom slučaju. MANOVA testom se utvrđuje da li postoji značajna razlika između ovih grupa, pri čemu se povoljnim ishodom smatra da razlika ne postoji. U Tabeli 5.17 su prikazani rezultati ovog testa. Pošto je u većini slučajeva $p >= 0.05$, zaključuje se da u pogledu strukturnih metrika ne postoje značajne razlike između isečaka koda kojima je dodeljen isti intenzitet. Odnosno, anotatori su međusobno konzistentni, što je povoljan ishod testa. Od ovih rezultata odstupaju dva slučaja – p-vrednost je manja od 0.05 za intenzitete 1 i 2 kod *dugačke metode*. Ovakvi rezultati ukazuju na nekonzistentnost anotatora prilikom dodeljivanja intenziteta 1 i 2 *dugačke metode*.

Tabela 5.17 Rezultati MANOVA testa pokazuju da, u pogledu strukturnih metrika, ne postoje značajne razlike između isečaka kojima je dodeljen isti intenzitet (gde je $p >= 0.05$).

Miris koda	**Intenzitet**			
	0	**1**	**2**	**3**
Dugačka metoda	Wilks' lambda = 0.997 $F(18,4181) = 0.796$ $p = 0.7072$	Wilks' lambda = 0.914 $F(18,1224) = 6.416$ $p = 0.000$	Wilks' lambda = 0.919 $F(18,479) = 2.35$ $p = 0.001$	Wilks' lambda = 0.885 $F(18,112) = 0.812$ $p = 0.683$
Velika klasa	Wilks' lambda = 0.988 $F(24,1472) = 0.752$ $p = 0.799$	Wilks' lambda = 0.936 $F(24,368) = 1.05$ $p = 0.397$	Wilks' lambda = 0.953 $F(24,149) = 0.304$ $p = 0.999$	Wilks' lambda = 0.797 $F(24,41) = 0.434$ $p = 0.983$

[29] Definicije strukturnih metrika su dostupne na https://github.com/Clean-CaDET/platform/blob/c4acff95ec00ff6c25fa62dde4818c1f40e39d39/CodeModel/CaDETModel/CodeItems/CaDETMetrics.cs.

ANOVA testom (engl. *Analysis of Variance*) [118] je ispitano koje pojedinačne strukturne metrike su dovedele do rezultata koji ukazuju na nekonzistentnost anotatora prilikom dodeljivanja intenziteta 1 i 2 *dugačke metode*. ANOVA test je upotrebljen za svaku metriku zasebno, a u Tabeli 5.18 su navedene metrike za koje je test rezultirao p-vrednostima manjim od 0.05. Rezultat ukazuje da ne postoji korelacija između strukturnih metrika i dodeljenog intenziteta, što je analizirano u Potpoglavlju 6.3 koje se bavi pretnjama validnosti.

Tabela 5.18 Strukturne metrike koje imaju najmanji uticaj na anotacije *dugačke metode*.

Intenzitet	Broj strukturnih metrika	Metrike
1	12 od 18	CYCLO, CYCLO_SWITCH, MLOC, MELOC, NOLV, MNOL, MNOC, MNOA, NONL, NOMO, MMNB, NOUW
2	10 od 18	CYCLO, CYCLO_SWITCH, MLOC, MELOC, NOLV, MNOL, MNOC, NOSL, MMNB, NOUW

Intenzitet mirisa koji anotatori dodeljuju se ne mora poklopiti sa izabranim skupom strukturnih metrika, jer metrike možda nisu relevantne za detekciju određenog mirisa koda ili je skup metrika nepotpun. Zapažanje da anotacije nisu u korelaciji sa strukturnim metrikama ne znači nužno da su anotacije nekvalitetne.

Za mirise *klasa podataka, zavist među odlikama* i *odbačeno nasledstvo* ne postoji široko prihvaćeni skup strukturnih metrika [14]. U Tabeli 3 u radu [14] se može videti velik broj pravila za detekciju koja koriste različite strukturne metrike za ove mirise koda. Za razliku od *dugačke metode* i *velike klase*, čija se pravila za detekciju oslanjaju na sličan skup metrika, pomenuta tri mirisa koda imaju najmanji nivo slaganja u pogledu strukturnih metrika korišćenih za njihovu detekciju [14].

ANOVA test je upotrebljen za *klasu podataka*, *zavist među odlikama* i *odbačeno nasledstvo*, kako bi se ispitala korelacija anotacija i strukturnih metrika. Rezultati se nalaze u Tabeli 5.19, gde su za svakog anotatora i svaki miris koda prikazane p-vrednosti. Naznačene su p-vrednosti < 0.05 koje ukazuju na značajnu korelaciju metrike i dodeljenog intenziteta mirisa. Nijedna strukturna metrika za *klasu podataka* nema značajnu korelaciju sa dodeljenim intenzitetima. U slučaju *zavisti među odlikama,* jedino AID metrika ima značajnu korelaciju sa dodeljenim intenzitetima. Za kraj, jedino metrika NMD_NAD (engl. *Number of Methods Declared and Number of Attributes Defined*) ima značajnu korelaciju sa dodeljenim intenzitetima za *odbačeno nasledstvo.*

Tabela 5.19 Rezultati ANOVA testa ukazuju na korelaciju strukturnih metrika i dodeljenih intenziteta mirisa. Za svaku strukturnu metriku i svakog anotatora su navedene p-vrednosti. P-vrednost manja od 0.05 ukazuje na značajnu korelaciju strukturne metrike i dodeljenih intenziteta.

Anotator	Klasa podataka					Zavist među odlikama			Odbačeno nasledstvo	
	WMC	LCOM	WOC	NOPA	NOPP	AID	ALD	NIC	NMD_NAD	BUR
1	0.61	0.34	0.85	0.68	0.36	**0.009**	0.46	0.84	**0.02**	0.05
2	0.51	0.82	0.53	0.08	0.96	**0.0009**	0.63	0.69	**0.003**	0.34
3	0.69	0.16	0.84	0.2	0.64	**0.000049**	0.25	0.2	**0.01**	0.39

Pored toga, rezultati ML modela obučenih na strukturnim metrikama pokazuju da metrike nisu u stanju da uhvate semantiku ovih mirisa. Škipina i saradnici [104] su objavili F-meru 0.62 za *klasu podataka* i 0.26 za *zavist među odlikama*. Slično, u radu [122] su autori prijavili F-meru 0.63 za *klasu podataka* i 0.32 za *zavist među odlikama*.

U Tabeli 5.20 su predstavljene performanse pravila zasnovanih na strukturnim metrikama za detekciju *klase podataka, zavisti među odlikama* i *odbačenog nasledstva*. Poređenja radi, u tabeli se takođe nalaze performanse pravila za detekciju *dugačke metode* i *velike klase*

objavljene u radu [71]. Za evaluaciju pravila korišćen je skup podataka koji je rezultat predloženog rešenja za ručno anotiranje mirisa[30].

Tabela 5.20 Performanse pravila za detekciju zasnovanih na strukturnih metrikama.

Miris koda	Pravilo za detekciju	F-mera
Klasa podataka	(WOC < 0.33) & (NOPA + NOPP > 4) & (WMC < 47 | NOPA + NOPP > 2) & (WMC < 31)	**0.19**
	(LCOM > 2) | (NOPP > 10)	0.09
	(WMC < 50) | (LCOM < 0.8)	0.11
	(WOC < 0.33) | (NOPA > 3) | (NOPP > 3)	0.12
Zavist među odlikama	(AID > 4) & (AID in top 10%) & (ALD < 3) & (NIC < 3)	**0.1**
Odbačeno nasledstvo	(BUR < 0.33) & (NMD + NAD > 2)	**0.34**
Velika klasa	(CLOC > 100) | (WMC > 20)	**0.83**
Dugačka metoda	(MLOC > 50) | (VG > 10)	**0.63**

Rezultati pokazuju da pravila zasnovana na strukturnim metrikama nemaju dobre performanse za *klasu podataka, zavist među odlikama* i *odbačeno nasledstvo.* Ove performanse su značajno lošije u odnosu na performanse postignute za *dugačku metodu* i *veliku klasu.* Na osnovu ovih rezultata se može potvrditi da *klasa podataka, zavist među odlikama* i *odbačeno nasledstvo* imaju najmanji nivo slaganja u pogledu odabranih strukturnih metrika za njihovu detekciju.

5.4.2 Ispunjenost zahteva 2

Drugi zahtev koji adresira fenomen nekonzistentnih anotacija definiše da svaki isečak koda mora biti anotiran od strane više anotatora. Ako bar dva anotatora anotiraju jedan isečak, anotacije se mogu validirati i prodiskutovati, što dovodi do pouzdanijih i konzistentnijih anotacija.

Anotiranje skupa podataka je jedna od aktivnosti anotacione procedure koja je projektovana tako da svaki isečak koda anotiraju bar dva anotatora (opisano u Potpoglavlju 4.2.4). Anotiranje isečaka koda od strane više anotatora je podržano konceptualnim modelom koji definiše *pojedinačne labele* koje se objedinjuju u *konačnu labelu* (opisano u Potpoglavlju 4.1). DSE alat omogućava istovremeno anotiranje isečaka koda od strane više anotatora i objedinjavanje pojedinačnih labela u konačnu labelu. Pored toga, alat identifikuje isečke koda koji nisu anotirani bar dva puta.

U Tabeli 5.21 se može videti da je ukupno anotirano 4154 isečaka koda, ali da je ukupan broj pojedinačnih labela 9708. Dakle, svaki isečak su anotirala bar dva anotatora.

Tabela 5.21 Broj anotiranih isečaka koda i pojedinačnih *labela* za svaki miris koda u skupu podataka.

	Dugačka metoda	Velika klasa	Klasa podataka	Zavist među odlikama	Odbačeno nasledstvo	Ukupno
Broj anotiranih isečaka koda	2574	920	231	220	209	4154
Broj pojedinačnih labela	6072	2130	491	549	466	9708

[30] Tabela 5.11 sadrži korisne resurse vezane za skup podataka.

5.4.3 Ispunjenost zahteva 3

Zahtev 3 adresira fenomen nekonzistentnih anotacija tako što traži da anotatori diskutuju i razreše neslaganja.

Ako konačnu labelu nije moguće formirati zbog konfliktnih pojedinačnih labela, aktivnost *anotiranja skupa podataka* (opisana u Potpoglavlju 4.2.4) definiše da ostali anotatori, koji prvobitno nisu anotirali isečak koda, anotiraju isečak i na taj način razreše postojeće konflikte. Ukoliko dodatno anotiranje ne razreši konflikte, *diskusija* je naredna aktivnost u anotacionoj proceduri gde se konflikti moraju razrešiti (opisano u Potpoglavlju 4.2.5). Razrešavanje nesuglasica je podržano konceptualnim modelom koji definiše *pojedinačne labele* koje sadrže *intenzitet* i *primenjene heuristike* (opisano u Potpoglavlju 4.1). DSE alat identifikuje konfliktne isečke koda koje treba da anotiraju anotatori koji ih prvobitno nisu anotirali. Nakon toga, identifikuje isečke koda čije konflikte treba razrešiti u okviru diskusije.

Svi konflikti nastali tokom kreiranja skupa podataka su razrešeni. U Tabeli 5.22 se može videti broj anotiranih isečaka koda i broj razrešenih konflikta za svaki miris koda. Anotatori su ukupno razrešili 1400 konflikata, što predstavlja oko 33% isečaka koda[31]. S obzirom na značajan procenat razrešenih konflikata, diskusija je značajno doprinela kvalitetu skupa podataka.

Tabela 5.22 Broj razrešenih konflikata za svaki miris koda i za ceo skup podataka.

Miris koda	Broj isečaka koda	Broj razrešenih konflikta
Dugačka metoda	2574	924
Velika klasa	920	290
Klasa podataka	231	29
Zavist među odlikama	220	109
Odbačeno nasledstvo	209	48
Ukupno	4154	1400

5.4.4 Ispunjenost zahteva 4

Po zahtevu 4, rešenje za ručno anotiranje mirisa koda bi trebalo da ubrza proces anotiranja, kako bi se mogli kreirati veliki skupovi podataka.

Aktivnost *odabir isečaka koda* (opisana u Potpoglavlju 4.2.2) omogućava uklanjanje trivijalnih isečaka kako bi se anotatori fokusirali na isečke koji zahtevaju dublju analizu. Zatim, aktivnost *anotiranja skupa podataka* (opisana u Potpoglavlju 4.2.4) propisuje da isečci koda budu podeljeni među anotatorima tako da svaki isečak prvo anotiraju dva anotatora, a ne svi. Konceptualni model podržava zahtev 4 tako što definiše *heuristike* koje treba da pojednostave anotiranje [59] i *svojstva koda* koja treba da usmere pažnju anotatora na relevantne informacije za svaki miris koda (opisano u Potpoglavlju 4.1).

DSE alat pruža funkcionalnost uklanjanja trivijalnih isečaka koda. Zatim, alat olakšava anotatorima analizu različitih svojstava koda, kao što su izvorni kod, strukturne metrike i veze među komponentama. Za prikaz veza među komponentama su upotrebljeni grafovi, gde svaki čvor predstavlja komponentu, a grana predstavlja vezu (poziv metode, nasleđivanje, itd.).

[31] Za razliku od IAA (Tabela 5.14) koji nam daje uvid u slaganje među anotatorima u konačnoj verziji skupa podataka, ovaj procenat nam ukazuje na broj konflikata nastalih tokom anotiranja.

Pored toga, alat prikazuje anotatorima jačinu veze (na primer, koliko puta je neka metoda pozvana) debljinom grane i težinama na čvorovima. Prikaz različitih svojstava koda na jednom mestu pojednostavljuje i ubrzava anotiranje, jer anotatori ne moraju da troše vreme na analizu svojstava koda na više mesta [100]. Zatim, alat omogućava anotatorima popunjavanje anotacione forme u okviru koje se nalazi spisak heuristika relevantnih za miris koda koji se anotira. Prikaz heuristika u okviru anotacione forme ubrzava i olakšava proces anotiranja, jer anotatori ne moraju samostalno da prave liste ili beleške. Pošto alat pruža uniformno beleženje anotacija za sve anotatore, anotacije je lakše obraditi i analizirati [100].

Zahtev 4 nije empirijski potvrđen zbog ograničenih resursa za istraživanje. Tačnije, nije postojala kontrolna grupa anotatora koja bi anotirala mirise bez primene predloženog rešenja, te nije moguće potvrditi da li je rešenje ubrzalo anotiranje. Nepostojanje kontrolne grupe je razmotreno u Potpoglavlju 6.3 koje se bavi pretnjama validnosti.

5.4.5 Ispunjenost zahteva 5

U petom zahtevu se navodi da odnos mirisa koda i čistog koda treba da oslikava njihov realan odnos u softverskim sistemima.

Aktivnost *odabir isečaka koda* (opisana u Potpoglavlju 4.2.2) propisuje nasumično biranje isečaka koda iz softverskih projekata. DSE alat automatizuje proces nasumičnog odabira isečaka, omogućavajući anotatorima da definišu broj ili procenat isečaka koje treba nasumično izvući iz projekta.

U Tabeli 5.23 se može videti da je odnos mirisa i čistog koda u skupu podataka nastalom primenom predložene anotacione procedure sličan odnosu u MLCQ skupu podataka [27], gde su autori takođe nasumično birali isečke koda. Konačne labele koje su prikazane u Tabeli 5.23 za MLCQ skup podataka su izračunate tehnikom većinskog glasanja (engl. *majority vote aggregation*)[32] primenjenog nad pojedinačnim labelama. Pošto autori MLCQ skupa nisu razrešavali konflikte, za pojedine isečke nije bilo moguće formirati konačnu labelu[33], te su uklonjeni iz analize u Tabeli 5.23. Pored toga, MLCQ skup podataka nema anotacije za miris *odbačeno nasledstvo*. Trenutno ne postoje drugi javno dostupni ručno anotirani skupovi podataka koji sadrže intenzitete za ovaj miris koda.

[32] U MLCQ skupu je više anotatora anotiralo isečke koda, ali nije svaki isečak anotiran od strane više anotatora. Za isečke koje je anotirao samo jedan anotator je konačna labela jednaka pojedinačnoj labeli koju je taj anotator dodelio.

[33] U slučaju *dugačke metode*, za 3 isečka nije bilo moguće izračunati konačnu labelu, jer su dodeljeni intenziteti bili različiti, a nijedan od njih nije bio dominantan. U slučaju *velike klase*, za 14 isečaka nije bilo moguće izračunati konačnu labelu.

Tabela 5.23 Poređenje konačnih labela u odnosu na MLCQ skup podataka *[27]*.

Skup podataka u ovom istraživanju						**MLCQ skup podataka**				
Miris koda	**Broj isečaka koda**	**Intenziteti u konačnim labelama**				**Broj isečaka koda**	**Intenziteti u konačnim labelama**			
		0	**1**	**2**	**3**		**0**	**1**	**2**	**3**
Dugačka metoda	2574	1924 (74.7%)	417 (16.2%)	182 (7.1%)	51 (2%)	2423	2148 (88.6%)	151 (6.3%)	98 (4%)	26 (1.1%)
Velika klasa	920	677 (73.6%)	145 (15.8%)	69 (7.5%)	29 (3.2%)	2312	2068 (89.4%)	157 (6.8%)	74 (3.2%)	13 (0.6%)
Klasa podataka	231	216 (93.5%)	12 (5.2%)	3 (1.3%)	0	2322	2031 (87.5%)	144 (6.2%)	118 (5%)	29 (1.2%)
Zavist među odlikama	220	184 (83.6%)	28 (12.7%)	8 (3.6%)	0	2406	2340 (97.3%)	47 (1.9%)	19 (0.8%)	0
Odbačeno nasledstvo	209	173 (82.7%)	35 (16.7%)	1 (0.5%)	0	/	/	/	/	/

5.4.6 Ispunjenost zahteva 6

Zahtev 6 traži od rešenja za ručno anotiranje da omogući anotiranje bilo kog Fowlerovog mirisa koda, kako bi se mogli kreirati skupovi podataka koji nemaju ograničenu pokrivenost mirisa koda.

Aktivnost *definisanje anotacione šeme i smernica* (opisana u Potpoglavlju 4.2.1) omogućava definisanje bilo kog Fowlerovog mirisa koda. Zatim, aktivnost *anotiranja skupa podataka* (opisana u Potpoglavlju 4.2.4) propisuje da se za svaki miris koda analiziraju svojstva koda i heuristike koje su relevantne za posmatrani miris. Zahtev 6 je podržan konceptualnim modelom koji definiše *anotacionu šemu* koja omogućava definisanje bilo kog Fowlerovog *mirisa koda* i relevantnih *heuristika* (opisano u Potpoglavlju 4.1). Zatim, *anotacione smernice* definišu *svojstva koda* relevantna za analizu specifičnog mirisa koda (opisano u Potpoglavlju 4.1). DSE alat omogućava definisanje i pregled anotacione šeme, pristup anotacionim smernicama i analizu različitih svojstava koda tokom anotiranja.

Tokom validacije, uspešno je anotirano 5 mirisa koda (uključujući „mirise unutar klasa“ i „mirise između klasa“ [45]), a tokom anotiranja su analizirana tri svojstva koda: izvorni kod, strukturne metrike i veze među komponentama. Rešenje je validirano na samo 5 mirisa, ali je u radu [61] demonstrirano da su svojstva koda koja su tokom validacije analizirana korisna i za anotiranje ostalih Fowlerovih mirisa.

5.5 Dokazivanje hipoteza

Nakon izvršenog eksperimenta u kom su anotatori anotirali skup podataka od pet mirisa koda, bilo je potrebno validirati ispunjenost zahteva definisanih za rešenje (Tabela 1.1 u Potpoglavlju 1.3). Zahtevi su definisani da adresiraju fenomene koji negativno utiču na ciljeve interesnih grupa. Rešenje koje ispunjava zahteve će doprineti ovim ciljevima, što je opisano argumentima doprinosa u Tabeli 1.1. U Potpoglavlju 1.3 je istaknuto i da je cilj istraživanja poravnat sa ciljevima interesnih grupa: unapređenje kvaliteta ručno anotiranih skupova podataka vezanih za mirise koda, što doprinosi kreiranju i obučavanju ML modela za detekciju mirisa, a zatim i poboljšanju održivosti koda, unapređenju kvaliteta softvera i smanjenju troškova razvoja softvera.

U Tabeli 5.24 se nalazi sumarizacija prethodnih potpoglavlja koji analiziraju ispunjenost zahteva. Istaknuti su koncepti i aktivnosti anotacione procedure koji doprinose ispunjavanju zahteva, a pored toga su navedene i funkcionalnosti DataSet Explorer alata koje doprinose ispunjavaju zahteva.

Tabela 5.24 Koncepti anotacione procedure i funkcionalnosti *DataSet Explorer* alata koji doprinose ispunjavanju zahteva definisanih za rešenje

Zahtev	**Koncepti i aktivnosti anotacione procedure koji ispunjavaju zahtev**	**Funkcionalnosti DSE alata koje ispunjavaju zahtev (opisane u Potpoglavlju 4.3.2)**
Z1: Rešenje treba da uključi obuku anotatora.	1. *Anotaciona šema* i *anotacione smernice* (opisane u Potpoglavlju 4.1) 2. *Obuka anotatora* (opisana u Potpoglavlju 4.2.3)	1. Pregled i definisanje anotacione šeme i smernica 2. Probno anotiranje 3. Identifikovanje konfliktnih isečaka koda koje treba prodiskutovati
Z2: Rešenje treba da podrži anotiranje od strane više anotatora.	1. *Konačna labela* objedinjuje p*ojedinačne labele* (opisano u Potpoglavlju 4.1) 2. *Anotiranje skupa podataka* (opisano u Potpoglavlju 4.2.4)	1. Istovremeno anotiranje isečaka koda od strane više anotatora 2. Provera da li je svaki isečak anotiran od strane bar dva anotatora 3. Objedinjavanje pojedinačnih labela u konačnu labelu
Z3: Rešenje treba da omogući razrešavanje nesuglasica među anotatorima, tj., konfliktnih anotacija.	1. *Pojedinačna labela* sadrži *primenjene heuristike* i *intenzitet* (opisano u Potpoglavlju 4.1) 2. *Anotiranje skupa podataka* (opisano u Potpoglavlju 4.2.4) 3. *Diskusija* (opisana u Potpoglavlju 4.2.5)	1. Identifikovanje konfliktnih isečaka koda koje treba da anotiraju anotatori koji ih prvobitno nisu anotirali 2. Identifikovanje konfliktnih isečaka koda koje treba prodiskutovati i razrešiti
Z4: Rešenje treba da ubrza anotiranje.	1. *Heuristike* i *svojstva koda* (opisane u Potpoglavlju 4.1) 2. *Odabir isečaka koda* (opisan u Potpoglavlju 4.2.2) 3. *Anotiranje skupa podataka* (opisano u Potpoglavlju 4.2.4)	1. Uklanjanje trivijalnih isečaka koda 2. Prikaz i analiza različitih svojstava koda 3. Prikaz heuristika za miris koda koji se anotira
Z5: Rešenje treba da omogući kreiranje skupova podataka sa realističnim odnosom mirisa koda i čistog koda.	1. *Odabir isečaka koda* (opisan u Potpoglavlju 4.2.2)	1. Nasumično biranje isečaka koda iz softverskih projekata
Z6: Rešenje treba da omogući anotiranje bilo kog mirisa koda definisanog od strane Martina Fowlera.	1. *Anotaciona šema* i *anotacione smernice* (opisane u Potpoglavlju 4.1) 2. *Definisanje anotacione šeme i smernica* (opisano u Potpoglavlju 4.2.1) 3. *Anotiranje skupa podataka* u okviru anotacione procedure (opisano u Potpoglavlju 4.2.4)	1. Definisanje anotacione šeme 2. Analiza različitih svojstava koda

Na osnovu ciljeva istraživanja je postavljena hipoteza da *rešenje za ručno anotiranje mirisa koda zasnovano na preskriptivnoj anotacionoj paradigmi* doprinosi većem kvalitetu skupova podataka u odnosu na druga postojeća rešenja. Dokazivanje ove hipoteze se svodi na dokazivanje podhipoteza na koje je razložena, a koje omogućavaju precizniju procenu da li rešenje povećava kvalitet skupa. Podhipoteze su potvrđene validiranje ispunjenosti relevantnih zahteva. U Tabeli 5.25 su navedene podhipoteze i relevantni zahtevi. Dodatno su navedeni argumenti doprinosa koji opisuju na koji način rešenje koje ispunjava zahtev doprinosi ostvarenju postavljenih ciljeva, a time i dokazuje postavljene podhipoteze.

Zbog ograničenih resursa za istraživanje i nepostojanja kontrolne grupe koja bi anotirala mirise bez primene predloženog rešenja, zahtev 4 nije empirijski potvrđen. Iako DSE alat pruža funkcionalnosti koje bi trebalo da ubrzaju i pojednostave anotiranje [100], a anotaciona procedura pojednostavljuje analizu mirisa njihovim razlaganjem na heuristike [59], ne može se garantovati ispunjenost zahteva 4 i potvrđivanje podhipoteze H2 bez poređenja vremena anotiranja kontrolne grupe i grupe anotatora koja prati predloženo rešenje. Ova pretnja validnosti je obrađena u Potpoglavlju 6.3.

Tabela 5.25 Dokazivanje hipoteza na osnovu validiranja ispunjenosti zahteva

Podhipoteza	Zahtev	Argument doprinosa
H1: Rešenje će povećati konzistentnost anotacija.	Z1: Rešenje treba da uključi obuku anotatora. Analiza ispunjenosti u Potpoglavlju 5.4.1.	Obuka anotatora usklađuje razumevanje anotatora o zadatku anotacije, čime se povećava njihova međusobna saglasnost i konzistentnost anotacija [38][39].
	Z2: Rešenje treba da podrži anotiranje od strane više anotatora. Analiza ispunjenosti u Potpoglavlju 5.4.2.	Kolaborativni pristup anotiranju smanjuje subjektivni uticaj i greške pojedinačnih anotatora, što dovodi do pouzdanijih i konzistentnijih anotacija [40][41].
	Z3: Rešenje treba da omogući razrešavanje nesuglasica među anotatorima, tj., konfliktnih anotacija. Analiza ispunjenosti u Potpoglavlju 5.4.3.	Razrešavanjem nesuglasica i ispravljanjem grešaka se povećava tačnost i konzistentnost anotacija [39][42].
H2: Rešenje će ubrzati anotiranje i omogućiti kreiranje većih skupova podataka.	Z4: Rešenje treba da ubrza anotiranje. Analiza ispunjenosti u Potpoglavlju 5.4.4.	Ubrzavanje procesa anotiranja omogućava kreiranje većih skupova podataka.
H3: Rešenje će omogućiti kreiranje skupova podataka koji oslikavaju realističan odnos koda koji sadrži mirise i koda koji ih ne sadrži.	Z5: Rešenje treba da omogući kreiranje skupova podataka sa realističnim odnosom mirisa koda i čistog koda. Analiza ispunjenosti u Potpoglavlju 5.4.5.	Odnos mirisa koda i čistog koda treba da predstavlja stvarnu distribuciju kako bi skupovi podataka bili korisni za obučavanje ML modela za detekciju mirisa [25].
H4: Rešenje će omogućiti kreiranje skupova podataka za sve vrste mirisa koda koje je definisao Martin Fowler.	Z6: Rešenje treba da omogući anotiranje bilo kog mirisa koda definisanog od strane Martina Fowlera. Analiza ispunjenosti u Potpoglavlju 5.4.6.	Skupovi podataka koji sadrže skup različitih mirisa koda su korisni za kreiranje robusnih ML modela za detekciju mirisa.

6. DISKUSIJA REZULTATA

Potpoglavlje 6.1 diskutuje o naučnim, razvojnim, aplikativnim i socio-ekonomskim doprinosima istraživanja. Diskusija o prednostima i ograničenjima preskriptivne anotacione paradigme na kojoj je zasnovano predloženo rešenje se nalazi u Potpoglavlju 6.2. Za kraj, u Potpoglavlju 6.3 su prodiskutovane pretnje validnosti.

6.1 Doprinosi istraživanja

Doprinosi istraživanja su razvrstani u četiri kategorije: naučni, razvojni, aplikativni i socio-ekonomski doprinos.

Ovo istraživanje ima tri naučna doprinosa. Prvo, projektovano je rešenje za ručno anotiranje mirisa koje je zasnovano na preskriptivnoj anotacionoj paradigmi [36]. Ova paradigma ograničava subjektivnost anotatora i sinhronizuje njihovo razumevanje. Time se smanjuje neslaganje među anotatorima i nekonzistentnost anotacija, a poboljšava kvalitet skupova i performanse ML modela obučenih na takvim skupovima. Iako anotiranje mirisa koda ima sličnosti sa semantičkom analizom teksta u NLP oblasti, gde se koristi preskriptivna anotaciona paradigma, postojeća istraživanja nisu isprobala takav pristup pri anotiranju mirisa koda. Na osnovu analize postojećih istraživanja, ovo istraživanje je prvo koje je usvojilo najbolje anotacione prakse iz NLP oblasti i primenilo ih u oblasti anotiranja mirisa.

Zatim, objavljene su anotacione šeme i smernice za mirise *dugačka metoda*, *velika klasa*, *klasa podataka*, *zavist među odlikama* i *odbačeno nasledstvo*. Ovo je posebno značajno za mirise *klasa podataka*, *zavist među odlikama* i *odbačeno nasledstvo*, za koje ne postoji velik broj empirijskih istraživanja o ručnom anotiranju ili analizi od strane inženjera. Pogotovo je izražen značaj u slučaju *odbačenog nasledstva*, čijim se ručnim anotiranjem bavilo samo jedno prethodno istraživanje koje nije objavilo anotacionu šemu i smernice [31]. Transparentne i detaljne anotacione šeme i smernice su korisne za izgradnju i obučavanje ML modela, jer omogućavaju dublju analizu grešaka modela. Primeri na kojima ML model greši, a čovek ne, se mogu analizirati kako bi se unapredile performanse modela [119], dok detaljne anotacione šeme i smernice omogućavaju analizu rezonovanja koje je dovelo do greške [119]. Iz tog razloga, anotacione šeme i smernice nastale u ovom istraživanju su javno dostupne.

Poslednji naučni doprinos se ogleda u objavljenom skupu podataka za mirise *dugačka metoda*, *velika klasa*, *klasa podataka*, *zavist među odlikama* i *odbačeno nasledstvo*. Postojeći skupovi podataka sadrže isečke koda napisane u Java programskom jeziku, dok ovaj skup sadrži C# isečke koda. Iako je C# zastupljen u diskusijama programera o mirisima koda [120], slabo je zastupljen u alatima za prepoznavanje [19][120]. Za svaki anotirani C# isečak koda, skup podataka sadrži finalnu anotaciju koja predstavlja odluku većine anotatora, pojedinačne anotacije svakog anotatora, vrednosti strukturnih metrika i heuristike označene kao primenjive od strane svakog anotatora. Pojedinačne anotacije anotatora omogućavaju analizu performansi anotatora, o čemu je bilo reči u Potpoglavlju 5.3. Vrednosti strukturnih metrika kvantifikuju različite aspekte koda, omogućavaju analizu konzistentnosti anotacija i testiranje različitih pristupa za prepoznavanje mirisa koda zasnovanih na strukturnih metrikama. Heuristike koje su anotatori označili kao primenjene daju uvid u način na koji su anotatori donosili odluke o anotaciji, što omogućava dublju analizu grešaka ML modela i poboljšanje njihovih performansi [119].

Razvojni doprinos se ogleda u projektovanju i implementaciji *DataSet Explorer* alata koji služi za anotiranje mirisa koda. Alat sadrži funkcionalnosti koje prate aktivnosti definisane u anotacionoj proceduri, te je razvijan sa idejom da ubrza i olakša rad anotatora tokom anotiranja. Alat je javno dostupan (korisni resursi navedeni su u Tabeli 4.1) i detaljno je opisan u Potpoglavlju 4.3.

Aplikativni doprinos se može analizirati kroz praktičnu primenu rešenja koje je projektovano da pomogne interesnim grupama u postizanju njihovih ciljeva. Aplikativni doprinos se ogleda u mogućnosti primene rešenja od strane anotatora kako bi kreirali skupove podataka visokog kvaliteta koji se mogu koristiti za obučavanje ML modela za prepoznavanje mirisa koda. Pored toga, aplikativni doprinos se ogleda i u mogućnosti upotrebe skupova podataka koji su rezultat predloženog rešenja od strane ML istraživača za obučavanje robusnih i nepristrasnih ML modela za prepoznavanje mirisa koda.

Na kraju, socio-ekonomski doprinos se ogleda u mogućnosti upotrebe ML modela za prepoznavanje mirisa koda obučenih na skupovima podataka koji su rezultat predloženog rešenja, od strane softverskih inženjera koji teže poboljšanju održivosti koda. Povećana održivost koda poboljšava kvalitet softvera, olakšava rad i smanjuje ukupne troškove razvoja softvera.

6.2 Prednosti i ograničenja preskriptivne anotacione paradigme

Interesne grupe treba da budu svesne prednosti i ograničenja skupova podataka koji su nastali kao rezultat procedure zasnovane na preskriptivnoj anotacionoj paradigmi.

Preskriptivna paradigma treba da smanji neslaganja među anotatorima i nekonzistentnost anotacija, čime se poboljšava kvalitet skupova i performanse ML modela [36]. Važno je naglasiti da preskriptivna paradigma podrazumeva da su anotatori obučeni da anotiraju u skladu sa unapred definisanim smernicama, kako bi kreirali skupove podataka sa konzistentnim anotacijama pogodnim za obučavanje ML modela. ML modeli obučeni na ovako kreiranim skupovima podataka možda ne odgovaraju interesnim grupama u specifičnim kontekstima[34], jer dosledno primenjuju smernice kojima su se anotatori vodili prilikom anotiranja. Različite interesne grupe bi mogle zahtevati različite anotacione smernice koje bi bile u skladu sa njihovim kontekstom. Zbog toga je veoma važno dokumentovati anotacione smernice kako bi se interesnim grupama pružio uvid u uverenja koja su ugrađena u skup podataka. U slučaju da se ta uverenja kose sa njihovim uverenjima, anotacione smernice se mogu prilagoditi interesnim grupama i njihovom kontekstu. Transparentnost anotacionih smernica je neophodna kako bi interesne grupe procenile da li se rezultujući skup podataka poklapa sa njihovim razumevanjem mirisa koda, te da li odgovara njihovim potrebama.

Sa druge strane, deskriptivna paradigma podržava subjektivnost anotatora u cilju prikupljanja različitih uverenja [36]. Za razliku od procedura koje su zasnovane na preskriptivnoj paradigmi, skupovi podataka koji su rezultat deskriptivnih anotacionih procedura se ne mogu tako lako koristiti za obučavanje ML modela [36][37]. Na primer, deskriptivnu paradigmu su usvojili Madeyski i Lewowski pri izgradnji MLCQ skupa podataka, kako bi prikupili subjektivna razumevanja programera o mirisima koda [27]. Kada se uporede slaganja anotatora u MLCQ skupu i skupu nastalom upotrebom procedure predložene u ovom istraživanju (Potpoglavlje 5.4.1), može se zapazati da preskriptivna paradigma značajno doprinosi slaganju anotatora za svaki ispitani miris koda (*dugačka metoda*, *velika klasa*, *klasa podataka* i *zavist među odlikama*).

[34] Kontekst se odnosi na konvencije specifične za projekat, industrijske standarde ili druge faktore koji utiču na primenljivost skupova podataka koji su rezultat preskriptivne anotacione procedure.

6.3 Pretnje validnosti rešenja

Konstruktna validnost se bavi slaganjem rezultata istraživanja sa konceptualnim modelom korišćenim u istraživanju. Razmotrene su sledeće pretnje validnosti:

- *Dvosmislene definicije mirisa koda:* Dvosmislene i neprecizne definicije mirisa koda mogu narušiti efikasnost predloženog rešenja, jer se anotatori mogu osloniti na subjektivno razumevanje mirisa što dovodi do nekonzistentnih anotacija [22]. Kako bi se otklonila ova pretnja, anotatori su se kroz obuku upoznali sa anotacionom šemom i smernicama koje sadrže definicije i primere mirisa koda (detaljnije u Potpoglavlju 4.2.3).
- *Anotaciona paradigma:* Preskriptivna anotaciona paradigma na kojoj se zasniva anotaciona procedura u ovom istraživanju može ograničiti anotatore u donošenju odluka. Strogo praćenje anotacionih smernica može doprineti netačnim anotacijama ukoliko smernice nisu definisane dovoljno precizno da uhvate semantiku koju anotatori treba da uoče. Pored toga, anotacione smernice mogu sadržati pristrasna ili preterano pojednostavljena uverenja koja će se oslikati u anotacijama. Pošto preskriptivna paradigma zahteva sinhronizovanje razumevanja anotatora, pojedini anotatori možda neće izražavati svoja istinska uverenja. Kako bi se otklonile navedene pretnje, rešenje predstavljeno u ovom istraživanju podstiče anotatore na diskusiju i razrešavanje nesuglasica, kako bi mogli izraziti svoja mišljenja i unaprediti date smernice (detaljnije u Potpoglavljima 4.2.3, 4.2.4 i 4.2.5).
- *Intenzitet mirisa koda:* Intenzitet mirisa koda je u ovom istraživanju tretiran kao diskretna kategorija. Međutim, pokušaj mapiranja intenziteta mirisa na diskretne kategorije može dovesti do preteranog pojednostavljivanja i preterano izraženog neslaganja među anotatorima. Na primer, intenzitet mirisa od 2.4 bi se zaokružio na broj 2, dok bi se intenzitet od 2.6 zaokružio na broj 3. Međutim, upotreba kontinualnih vrednosti za intenzitet mirisa bi doprinela vremenskoj zahtevnosti ručnog anotiranja.
- *Podhipoteze i zahtevi definisani za rešenje:* Hipoteza ovog istraživanja je da *rešenje za ručno anotiranje mirisa koda zasnovano na preskriptivnoj anotacionoj paradigmi* doprinosi većem kvalitetu skupova podataka u poređenju sa postojećim rešenjima. Kako bi se preciznije procenilo da li rešenje povećava kvalitet skupa, definisane su četiri podhipoteze (Potpoglavlje 1.3). Zatim je definisano šest zahteva koje rešenje treba da ispuni kako bi rezultujući skup podataka bio kvalitetan (Potpoglavlje 1.3, Tabela 1.1). Moguće je da skup podhipoteza i zahteva nije potpun.

  Međutim, podhipoteze i zahtevi su definisani na osnovu prethodnih istraživanja i iskustava inženjera. Potpoglavlje 1.1 daje uvid u postojeće pristupe za prepoznavanje mirisa i probleme vezane za kvalitet skupova podataka. U Potpoglavlju 2.2 su razmotreni izazovi ručnog anotiranja, dok Potpoglavlja 2.5 i 2.6 analiziraju postojeće anotacione procedure i alate za anotiranje. Na osnovu ove analize su izdvojeni najčešći fenomeni koji narušavaju kvalitet skupova podataka (Potpoglavlje 1.2). Podhipoteze i zahtevi su definisani tako da adresiraju negativne fenomene, te postoji uverenje da će rešenje koje ispunjava zahteve doprineti povećanju kvaliteta skupova podataka, čime se potvrđuju definisane podhipoteze. Pošto je istraživanje zasnovano na *design science* metodologiji [70] koja definiše iterativno rešavanje problema, podržava se ideja da bi dodatne analize mogle dovesti do proširenja zahteva i podhipoteza.
- *Odabir svojstava koda:* Za analizu Fowlerovih mirisa koda je identifikovano pet relevantnih svojstava koda: izvorni kod, strukturne metrike, apstraktno sintaksno stablo (AST), istorija promena i veze među komponentama sistema [61]. Potpoglavlje 2.3 opisuje identifikovana svojstva. Međutim, za anotiranje pet odabranih mirisa u ovom istraživanju

su anotatori analizirali samo tri svojstva: izvorni kod, strukturne metrike i veze među komponentama sistema. Isključivanje istorije promena ili AST-a iz analize može dovesti do neprecizne anotacije određenih mirisa koda, za koje bi ova svojstva dala relevantne informacije. Ipak, postojeća istraživanja pokazuju da nisu sva svojstva veoma korisna za ručno anotiranje mirisa. Imajući u vidu vremensku zahtevnost ručnog anotiranja, odabrana su svojstva za koje postoji više istraživanja koja ukazuju na njihovu relevantnost za anotiranje mirisa.

Izvorni kod i strukturne metrike se često koriste za analizu Fowlerovih mirisa [14][23][45][57]. Fowler je istakao da programeri mogu tražiti strukture u izvornom kodu koje ukazuju na potrebu za refaktorisanjem i uklanjanjem mirisa [8], dok su empirijska istraživanja pokazala da programeri zaista koriste izvorni kod da analiziraju većinu Fowlerovih mirisa [23][57]. Sa druge strane, alati za detekciju mirisa zasnovani na strukturnim metrikama vrše detekciju svih Fowlerovih mirisa, sem *nepotpunih klasa u biblioteci*. I za kraj, iako je analiza veza među komponentama sistema slabo zastupljena u literaturi, doneta je odluka da se svojstvo uključi u istraživanje, jer je korisno za analizu „mirisa između klasa“ [45], kao što su *klasa podataka*, *zavist među odlikama* i *odbačeno nasledstvo*.

Sa druge strane, AST i istorija promena su se pokazali korisnim za automatsku detekciju mirisa [61]. Različite tehnike zasnovane na modelima koriste AST za otkrivanje određenih obrazaca u kodu. Međutim, empirijska istraživanja nisu pokazala da anotatori koriste AST za analiziranje mirisa koda. Iz tog razloga, ovo svojstvo nije integrisano u predloženo rešenje. Sa druge strane, Palomba i saradnici [29] su pokazali da istorija promena može doprineti automatskoj detekciji specifičnih mirisa, kao što su *divergentne izmene*, *operacija sačmarom* i *paralelne hijerarhije nasleđivanja*. Dobre performanse su postignute i prilikom detekcije mirisa *velika klasa* i *zavist među odlikama*. Međutim, nije evaluirano koliko je istorija promena korisna pri detekciji mirisa *klasa podataka* i *odbačeno nasledstvo*. Zbog nedostatka podataka za ova dva mirisa koda i vremenski zahtevne analize istorije promena, predloženo rešenje ne propisuje analizu ovog svojstva koda.

Pretnje *internoj validnosti* se tiču internih faktora koji mogu uticati na rezultate istraživanja:

- *Greške pri anotiranju:* Umorni ili nepažljivi anotatori mogu napraviti nenamerne greške tokom anotiranja mirisa koda. Greške se odnose na previde napravljene tokom analize mirisa ili pogrešno zabeležene anotacije, te dovode do nekonzistentnih anotacija, čime se smanjuje kvalitet skupova podataka. Kako bi se smanjio broj grešaka koje nastaju tokom ručnog anotiranja, svaki isečak koda su za početak anotirala dva anotatora. Ukoliko su njihove anotacije konfliktne, nije moguće kreirati konačnu labelu za anotirani isečak, te je bilo neophodno uključiti trećeg anotatora u anotiranje konfliktnog isečka. Ako bi konflikt opstao nakon treće anotacije, anotatori bi diskutovali isečak koda, pronašli razlog konflikta i ispravili pogrešne anotacije.

  Ukoliko je napravljen previd tokom analize mirisa, anotatori treba da konsultuju anotacione šeme i smernice koje su analizirali tokom obuke anotatora. Ovo može rezultovati poboljšanjem razumevanja mirisa ili unapređenjem anotacionih šema i smernica. Sa druge strane, ako se greška ogleda u pogrešno zabeleženoj anotaciji, tada nema potrebe za izmenom šema i smernica, već je potrebno samo ispraviti pogrešnu anotaciju.

  Pošto greške mogu narušiti konzistentnost anotacija u skupu podataka, izračunato je slaganje među anotatorima (engl. *Inter-Annotator Agreement* – IAA). Rezultati slaganja se nalaze u Potpoglavlju 5.4.1, te pokazuju da su anotatori bili konzistentni tokom anotiranja svih pet mirisa (*dugačke metode*, *velike klase*, *klase podataka*, *zavisti među odlikama* i

odbačenog nasledstva). Ovi rezultati ukazuju da je broj grešaka prilikom anotiranja sveden na minimum, te da je anotaciona procedura pouzdana.

- *Nedostatak MANOVA rezultata*: MANOVA statistički test je primenjen kako bi se procenila konzistentnost anotacija u pogledu strukturnih metrika za *dugačku metodu* i *veliku klasu*. Međutim, konzistentnost anotacija za *klasu podataka*, *zavist među odlikama* i *odbačeno nasledstvo* nije procenjena na isti način, zbog nerelevantnosti strukturnih metrika za detekciju ovih mirisa. Umesto toga je izračunato slaganje među anotatorima (IAA) pomoću statističke mere za procenu pouzdanosti i slaganja anotatora - *Krippendorff* alfa vrednost [115]. Detaljno objašnjenje upotrebe MANOVA testa i izračunavanja slaganja među anotatorima je u Potpoglavlju 5.4.1.
- *Nekonzistentnost pri anotiranju „dugačke metode"*: Rezultati MANOVA testa ukazuju na nekonzistentnost anotatora pri anotiranju *dugačke metode* intenzitetima 1 i 2 (Potpoglavlje 5.4.1). Pored primene ANOVA testa kojim se utvrdilo koje pojedinačne strukturne metrike nisu u korelaciji sa dodeljenim intenzitetom (Potpoglavlje 5.4.1), pregledani su isečci koda kojima je dodeljen isti intenzitet (1 ili 2), a imaju različite vrednosti strukturnih metrika izlistanih u Tabeli 5.18. Kod nekih isečaka su anotatori načinili slučajne greške, koje su zatim ispravljene. Međutim, postoje isečci kojima je dodeljen intenzitet 1 ili 2, iako metrike sugerišu da miris koda nije prisutan. Na primer, metode `EffectsExtension`[35] i `RegionCaptureTransparentForm`[36] nisu preterano dugačke i nemaju kompleksne strukture u kodu, ali su anotatori odlučili da im dodele intenzitet 1 zbog korišćenih naziva koji unose semantičku kompleksnost. Može se dati i suprotan primer, gde strukturne metrike ukazuju na postojanje mirisa (na primer, metoda ima značajan broj ugnježdenih i kompleksnih iskaza), ali korišćenje preciznih naziva utiče na percepciju anotatora o kompleksnosti, te je on tumači suprotno onome na šta metrike ukazuju.

  Prema tome, strukturne metrike ne mogu obuhvatiti sve aspekte koje anotatori razmatraju dok anotiraju mirise. Moguće je da skup strukturnih metrika korišćen u MANOVA testu ne može da doprinese razumevanju semantike koda. Fowler je istakao da nijedan skup metrika ne može da se meri sa intuicijom čoveka [11], čime se ljudska evaluacija smatra najmerodavnijom pri detekciji mirisa koda.
- *Loše performanse ML modela pri prepoznavanju „klase podataka" i „zavisti među odlikama"*: Iako su anotacije nastale u ovom istraživanju za ova dva mirisa konzistentnije od MLCQ skupa [27], performanse ML modela su slične performansama modela koji koriste MLCQ skup (Potpoglavlje 5.3). Problem je mali skup podataka koji sadrži anotacije za *klasu podataka* i *zavist među odlikama*. Tokom validacija rešenja, anotatori su anotirali samo ~200 isečaka koda za ova dva mirisa, pošto je njihovo anotiranje izuzetno vremenski zahtevno zbog kompleksnije analize u odnosu na *dugačku metodu* i *veliku klasu*.

  Iako performanse ML modela nisu bolje od performansi prethodnih istraživanja, konzistentno anotiran skup podataka sa pratećim anotacionim šemama i smernicama je

35 https://github.com/MonoGame/MonoGame/blob/4802d00db04dc7aa5fe07cd2d908f9a4b090a4fd/MonoGame.Framework/Platform/Audio/OpenAL.cs#L762-L785

36 https://github.com/ShareX/ShareX/blob/c9a71ed00eda0e7c5a45237b9bcd3f8f614cda63/ShareX.ScreenCaptureLib/Forms/RegionCaptureTransparentForm.cs#L64-L94

značajan doprinos ovog istraživanja, imajući u vidu nedostatak skupova podataka za pomenute mirise koda.

Vremenska zahtevnost ručnog anotiranja se može umanjiti ukoliko se anotiranje podeli među anotatorima na optimalan način. Veći broj anotatora, pri čemu je svaki isečak koda anotiran od strane dva anotatora, će učiniti anotiranje bržim, jer će svaki anotator anotirati manji procenat isečaka koda. Zatim, anotatori će biti efikasniji u anotiranju ako koriste adekvatne alate koji im pomažu u procesu anotiranja. Alati koji uklanjaju trivijalne isečke koda omogućavaju anotatorima da se fokusiraju na isečke koji zahtevaju dublju analizu. Alati koji na jednom mestu prikazuju isečke koda, svojstva koda i heuristike neophodne za detekciju mirisa će pomoći anotatorima da uštede vreme i trud, eliminišući potrebu da koriste spolje resurse prilikom anotiranja. Alati koji, pored analize isečaka koda, pružaju i beleženje anotacija na istom mestu takođe ubrzavaju proces anotiranja, uklanjajući potrebu da anotator menja različite alate i dokumente dok analizira isečke i beleži anotacije. DSE alat[37] predložen u ovom istraživanju nudi pomenute funkcionalnosti. Video tutorijali koji demonstriraju ove funkcionalnosti su dostupni online[38].

- *Nedostatak poređenja rezultata za „odbačeno nasledstvo“*: Za mirise *dugačka metoda*, *velika klasa*, *klasa podataka* i *zavist među odlikama* je bilo moguće napraviti poređenje slaganja među anotatorima (IAA) sa MLCQ skupom podataka [27]. Međutim, za *odbačeno nasledstvo* nije pronađen ručno anotiran skup podatak sa kojim bi se moglo izvršiti poređenje. Nedostatak skupova podataka koji sadrže *odbačeno nasledstvo* ograničava sposobnost zaključivanja da li je predloženo rešenje pouzdano za anotiranje ovog mirisa. Ipak, slaganje među anotatorima (IAA) je izračunato za *odbačeno nasledstvo*, kako bi se budućim istraživanjima omogućilo poređenje rezultata.
- *Subjektivnost anotatora:* Anotaciona šema i smernice koje su definisane u ovom istraživanju su prilagođene uverenjima anotatora koji su anotirali skup podataka. Kako bi se ograničila subjektivnost anotatora u šemama i smernicama, analizirana su empirijska istraživanja kako bi se utvrdile heuristike, svojstva koda i smernice za analizu mirisa koda. Kao rezultat detaljne analize postojeće literature, pretpostavlja se da je perspektiva ovog istraživanja u skladu sa perspektivom zajednice. Međutim, nedostatak empirijskih istraživanja za tri mirisa *(klasa podataka, zavist među odlikama* i *odbačeno nasledstvo*) otežava analizu praktičnih primera, tj., analizu isečaka koda koji su zahvaćeni mirisima različitog intenziteta. U ovom istraživanju su dokumentovane šeme i smernice kako bi se omogućila transparentnost za interesne grupe koje bi koristile rezultujući skup podataka. Transparentnost istraživanja je neophodna kako bi interesne grupe procenile da li se njihova uverenja poklapaju sa uverenjima ovog istraživanja, te da li skup podataka odgovara njihovim potrebama.

  Ukoliko se uverenja ne poklapaju, interesne grupe mogu izmeniti anotacione šeme i smernice. Nažalost, većina istraživanja ne vodi računa o transparentnosti [123][124], zbog čega se rezultati ne mogu ponoviti. Sistematski pregled literature [17] ističe da postojeći skupovi podataka nisu standardizovani, zbog čega je otežano poređenje skupova. Suprotno tome, ovo istraživanje je posebnu pažnju posvetilo dokumentovanju čitavog rada kako bi se omogućila ponovljivost rezultata. S obzirom na sveobuhvatnu analizu postojeće

[37] U Tabeli 4.1 se nalaze relevantni resursi za upotrebu DSE alata za anotiranje.

[38] https://www.youtube.com/playlist?list=PLOXMk-hlotHGGoyNgCEk285CQkUEUNCAW

literature, postoji uverenje da će većina programera smatrati rezultate ovog istraživanja korisnim.

Eksterna validnost se bavi generalizacijom rezultata istraživanja. Razmotrene su sledeće pretnje:

- *Programski jezik anotiranih isečaka koda:* Tokom validacije rešenja, anotatori su anotirali isečke koda napisane u C# programskom jeziku. Iako je rešenje projektovano za C#, neki delovi se mogu iskoristiti za anotiranje isečaka napisanih u drugom programskom jeziku.

  Rešenje se sastoji od anotacione procedure i alata za anotiranje. Anotaciona procedura ne zavisi od programskog jezika, već nalaže koje aktivnosti anotatori treba da prate kako bi anotirali skup podataka. Procedura nalaže analizu svojstava koda kao što su strukturne metrike, izvorni kod i veze među komponentama sistema. Ova svojstva se mogu menjati po potrebi, tj., mogu se odabrati druga svojstva koja su relevantna za analizu određenog mirisa ili isečka koda napisanog u drugom programskom jeziku. Veze među komponentama sistema podrazumevaju veze među klasama i metodama, te je ovo svojstvo relevantno ukoliko je u pitanju objektno-orijentisan programski jezik. Sa druge strane, alat za anotiranje je implementiran da omogući anotiranje C# isečaka koda, te bi se morao izmeniti kako bi se omogućilo anotiranje isečaka napisanih u drugim programskim jezicima.

- *Odabrani isečci koda:* Tokom validacije rešenja su anotirani isečci koda koji su odabrani iz relativno malog skupa softvera. Za *dugačku metodu* i *veliku klasu* su isečci odabrani iz 8 softvera, za *klasu podataka* i *odbačeno nasledstvo* iz 15 softvera, a za *zavist među odlikama* iz 11 softvera.

  Kako bi se umanjio negativan uticaj ove pretnje, isečci koda su birani nasumično iz softvera sa različitim stilovima kodiranja, praksama i kontekstima (poput video igara, aplikacija za upravljanje datotekama i različitih biblioteka). Pošto ispoljavanje i prepoznavanje mirisa koda zavisi od konteksta, obrazaca i stilova primenjenih pri pisanju koda, moguće je da odabrani softveri nisu dovoljni da se rezultati istraživanja generalizuju na druge tipove softvera.

- *Ograničen broj anotatora:* Anotiranje skupa podataka u sklopu validacije rešenja su izvršila tri anotatora, pri čemu može biti izražena pristrasnost i subjektivna interpretacija anotacione procedure. Ova pretnja se razrešava obučavanjem anotatora kako bi se osiguralo razumevanje anotacione procedure, kao i diskusijom u kojoj anotatori mogu izneti svoja zapažanja. Grupa od tri anotatora ne može imati perspektivu veće grupe anotatora i potencijalno se ne susreće sa izazovima sa kojima bi se susrela veća grupa anotatora. Zbog toga, grupa je sastavljena od anotatora sa različitim iskustvom. U vreme validacije rešenja je prvi anotator bio profesor na četiri predmeta iz oblasti softverskog inženjerstva, sa šest godina iskustva u industriji. Preostali anotatori su studenti doktorskih studija koji se bave istraživanjem održivosti softvera i imaju dve godine iskustva na manjim projektima. Međutim, grupa anotatora blisko sarađuje u zajedničkom okruženju, te bi uključivanje anotatora iz drugih okruženja donelo dodatne uvide.

  Kreiranje veće grupe anotatora je izazovno zato što anotiranje mirisa koda zahteva specifičnu ekspertizu, kako bi anotatori mogli razumeti dizajn i semantiku koda. Pored znanja o razvoju softvera, arhitekturi softvera, principima dizajna i programskim jezicima, anotatori moraju posedovati duboko razumevanje o mirisima koda kako bi ih mogli efektivno anotirati. Duboko razumevanje mirisa koda omogućava anotatorima da prevaziđu

površno posmatranje koda i identifikuju suptilne obrasce u kodu koji upućuju na prisustvo mirisa. Specifično znanje o mirisima čini anotatore sposobnijim za pravljenje preciznijih i značajnijih anotacija. Formiranje grupe anotatora koju čine eksperti u oblasti mirisa koda zahteva značajno finansijsko ulaganje zbog vrednosti njihovog specifičnog znanja i vremena koje treba da ulože u anotiranje.

Obučavanje anotatora koji imaju osnovno znanje o softverskom inženjerstvo bi moglo smanjiti troškove formiranja grupe anotatora koju čine stručnjaci u oblasti mirisa koda. Ovaj pristup bi omogućio formiranje većih grupa anotatora, kao i lakše pronalaženje zamenskih anotatora ukoliko je to potrebno. Zamena stručnjaka anotatorima sa manje iskustva je izvodljiva pošto prethodna istraživanja pokazuju da iskustvo anotatora i iskustvo u industriji ne doprinose većem slaganju anotatora, tj., kvalitetnijim anotacijama [21][101]. Rešenje predloženo u ovom istraživanju ne zahteva anotiranje od strane stručnjaka u oblasti mirisa koda, već propisuje obučavanje anotatora koji imaju osnovno znanje iz oblasti softverskog inženjerstva. Ovo je izuzetno relevantno u kontekstu industrije, gde bi stručnjaci sa više iskustva mogli definisati anotacionu šemu i smernice, a manje iskusni programeri bi prošli kroz obuku neophodnu za anotiranje. Na ovaj način se smanjuju troškovi potrebni za kreiranje kvalitetnog skupa podataka, pošto stručnjaci ne obavljaju deo posla koji je izuzetno vremenski zahtevan.

- *Nedostatak industrijskog konteksta:* Iako anotatori imaju određeno iskustvo, oni nisu primarno zaposleni kao softverski inženjeri u industriji. Predloženo rešenje nije validirano u industrijskom kontekstu sa inženjerima kojima je to primarno radno mesto. Ovo potencijalno ne predstavlja problem, pošto prethodno istraživanje pokazuje da iskustvo u radu u industriji nije povezano sa načinom anotiranja [101], te da ne postoje dokazi da industrijski kontekst dovodi do većeg slaganja anotatora [21]. Ipak, bilo bi korisno proširiti grupu anotatora kako bi se potvrdila prethodna istraživanja.

Validnost zaključivanja se bavi ispravnošću zaključaka izvedenih iz rezultata istraživanja. Razmotrena je sledeća pretnja:

- *Nedostatak kontrolne grupe:* U sklopu validacije rešenja nije postojala kontrolna grupa anotatora koja bi anotirala mirise koda bez praćenja predložene anotacione procedure i bez korišćenja DSE alata. Takođe, nije postojala kontrolna grupa koja bi primenila neku od postojećih anotacionih procedura obrađenih u Potpoglavlju 2.5. Razlog nepostojanja kontrolne grupe se ogleda u ograničenim resursima za istraživanje. Ručno anotiranje je kompleksno i vremenski zahtevno, te nisu postojali uslovi za sprovođenje ovog eksperimenta. Nepostojanje kontrolne grupe otežava evaluaciju zahteva Z4 da rešenje treba da ubrza anotiranje, pošto se ne može uporediti vreme i efikasnost sa grupom anotatora koja je koristila predloženo rešenje. Stoga, podhipoteza H2 nije empirijski potvrđena (H2: Rešenje će ubrzati anotiranje i omogućiti kreiranje većih skupova podataka).

  Međutim, zapažanja anotatora prikupljena tokom anotiranja skupa podataka ukazuju da je anotiranje efikasnije kada se na jednom mestu nalaze sva svojstva koda koja je potrebno analizirati. Takođe, anotatori ističu lakše i brže anotiranje kada se analiza mirisa svede na analizu heuristika. Kako druga istraživanja navode da upotreba heuristika pojednostavljuje anotiranje [59], a funkcionalnosti alata mogu ubrzati anotiranje [100], postoji pretpostavka da predloženo rešenje ispunjava zahtev Z4 i hipotezu H2.

- *Ograničen broj isečaka koda:* Prilikom validacije rešenja, anotatori su anotirali ukupno 660 isečaka koda za mirise *klasa podataka*, *zavist među odlikama* i *odbačeno nasledstvo*, dok

anotirani broj isečaka za *dugačku metodu* i *veliku klasu* iznosi 3494. Iako hipoteza H2 navodi da će rešenje omogućiti kreiranje velikih skupova podataka, broj isečaka koda za *klasu podataka*, *zavist među odlikama* i *odbačeno nasledstvo* nije velik, zbog ograničenih resursa, tj., broja anotatora i dostupnog vremena.

Razlika u broju anotiranih isečaka po mirisu je prisutna zato što analiza različitih mirisa koda ima različitu vremensku zahtevnost i kompleksnost. *Dugačka metoda* i *velika klasa* su „mirisi unutar klasa" za koje su anotatori posmatrali izolovani isečak koda koji sadrži metodu ili klasu koja se anotira. Sa druge strane, ostali anotirani mirisi koda su „mirisi između klasa" za koje je neophodno analizirati dizajn i semantiku sistema (detaljnije u Potpoglavlju 2.2).

U poređenju sa brojem anotiranih isečaka koda u drugim istraživanjima, broj anotiranih isečaka u ovom istraživanju je relativno značajan. Na primer, u ovom istraživanju su tri anotatora anotirala 231 isečak za *klasu podataka*, a 220 za *zavist među odlikama*. U MLCQ skupu [27] je 26 anotatora anotiralo 2322 isečka za *klasu podataka* i 2406 za *zavist među odlikama*, iz čega se vidi da razlika u prosečnom broju isečaka po anotatoru nije značajna. Međutim, za razliku od MLCQ skupa, anotatori u ovom istraživanju su prošli kroz obuku, razrešili su svaku konfliktnu anotaciju, i bar dva anotatora su anotirala svaki isečak koda.

- *Ograničen broj mirisa koda:* Validacija rešenja na malom broju mirisa koda može uticati na generalizaciju rešenja na druge mirise, čime je otežano evaluiranje zahteva Z6 da rešenje treba da omogući anotiranje bilo kog Fowlerovog mirisa. Stoga, hipoteza H4 nije u potpunosti dokazana (H4: Rešenje će omogućiti kreiranje skupova podataka za sve vrste mirisa koda koje je definisao Martin Fowler). Predloženo rešenje je validirano na pet mirisa (*velika klasa*, *dugačka metoda*, *klasa podataka*, *zavist među odlikama* i *odbačeno nasledstvo*), čiji je odabir obrazložen u Potpoglavlju 2.1. U nastavku teksta su analizirane kategorije kojima ovi mirisi pripadaju i svojstva koda koja su relevantna za njihovo anotiranje. Na osnovu te analize se može obrazložiti evaluiranje zahteva Z6 i dokazivanje hipoteze H4. Kategorije koje će biti razmotrene su definisali Mantyla i saradnici [46] i Lacerda i saradnici [45].

  Velika klasa i *dugačka metoda* pripadaju kategoriji *bloaters,* koja sadrži mirise koji predstavljaju velike komponente koda koje je teško razumeti i menjati [46]. Empirijska istraživanja pokazuju da anotatori koriste izvorni kod da analiziraju mirise koda iz kategorije *bloaters* [61]. Predloženo rešenje propisuje analizu izvornog koda, te se smatra da omogućava i anotiranje ostalih mirisa iz iste kategorije. Za svaki pojedinačan miris koda je moguće definisati heuristike kroz anotacionu šemu i smernice.

  Klasa podataka se nalazi u kategoriji *dispensables*, koja predstavlja nepotreban kod koji treba ukloniti ili ga učiniti korisnijim dodavanjem odgovornosti [46]. *Zavist među odlikama* pripada kategoriji *couplers,* predstavljajući visok stepen spregnutosti što se kosi sa objektno-orijentisanim principima [46]. *Odbačeno nasledstvo* je u kategoriji *object-oriented abusers,* koja ukazuje na nepravilnu upotrebu objektno-orijentisanih principa [46]. Empirijska istraživanja pokazuju da anotatori koriste izvorni kod prilikom analize mirisa iz navedenih kategorija. Pored toga, većina mirisa iz navedenih kategorija su „mirisi između klasa" za koje je potrebna analiza dizajna sistema [45]. Pošto predloženo rešenje omogućava anotatorima analizu izvornog koda i veza među komponentama sistema, smatra se da rešenje omogućava i anotiranje ostalih mirisa iz navedenih kategorija.

  U ovom istraživanju nisu anotirani mirisi iz kategorija *change preventers, encapsulators* i *others* [46]. Međutim, empirijska istraživanja pokazuju da programeri koriste izvorni kod da analiziraju mirise iz ovih kategorija [61]. Pored toga, većina mirisa iz navedenih

kategorija su „mirisi između klasa“ za koje je potrebno analizirati dizajn sistema, što rešenje omogućava kroz analizu veza među komponentama sistema. U budućnosti, validacija rešenja na ovim mirisima koda bi omogućila kompletnije evaluiranje ispunjenosti zahteva da rešenje omogući anotiranje bilo kog Fowlerovog mirisa.

7. ZAKLJUČAK

Cilj istraživanja je unapređenje kvaliteta ručno anotiranih skupova podataka vezanih za mirise koda. Kako bi se postigao navedeni cilj, projektovano je rešenje za ručno anotiranje mirisa koda koje je zasnovano na preskriptivnoj anotacionoj paradigmi. Rešenje obuhvata anotacionu proceduru i alat za anotiranje, a projektovano je na osnovu zahteva koji adresiraju negativne fenomene koji narušavaju kvalitet skupova. Fenomeni uključuju nekonzistentne anotacije, mali broj anotacija, nerealističan odnos mirisa koda i čistog koda i ograničenu pokrivenost mirisa koda. Spram toga, u zahtevima se navodi da rešenje treba da uključi obuku anotatora, anotiranje od strane više anotatora i razrešavanje konfliktnih anotacija. Zatim, zahteva se da rešenje ubrza anotiranje i omogući kreiranje skupa sa realističnim odnosom mirisa i čistog koda, te anotiranje bilo kog Fowlerovog mirisa.

Rešenje je validirano u eksperimentu tokom kog je grupa od tri anotatora pratila anotacionu proceduru i koristila alat za anotiranje skupa podataka. Grupa je anotirala pet mirisa koda: *dugačka metoda*, *velika klasa*, *klasa podataka*, *zavist među odlikama* i *odbačeno nasledstvo*. Na osnovu sprovedenog eksperimenta je analizirana ispunjenost zahteva definisanih za rešenje, nakon čega je usledilo dokazivanje postavljenih hipoteza istraživanja.

Naučni doprinos istraživanja uključuje:

- Rešenje za ručno anotiranje mirisa zasnovano na preskriptivnoj anotacionoj paradigmi, koja ograničava subjektivnost anotatora i smanjuje nekonzistentnost anotacija. Na osnovu analize postojećih istraživanja, ovo je prvo rešenje koje je usvojilo najbolje anotacione prakse iz NLP oblasti i primenilo ih u oblasti mirisa koda.
- Anotacione šeme i smernice za pet mirisa koda: *dugačka metoda*, *velika klasa*, *klasa podataka*, *zavist među odlikama* i *odbačeno nasledstvo*. Ovo je posebno značajno za poslednja tri mirisa za koje ne postoji velik broj empirijskih istraživanja o ručnom anotiranju.
- Skup podataka za pet mirisa koda: *dugačka metoda*, *velika klasa*, *klasa podataka*, *zavist među odlikama* i *odbačeno nasledstvo*.

Razvojni doprinos uključuje projektovanje i implementaciju *DataSet Explorer* alata koji služi za anotiranje mirisa koda.

Aplikativni doprinos se ogleda u mogućnosti primene rešenja od strane anotatora za kreiranje kvalitetnih skupova podataka pomoću kojih se obučavaju ML modeli za prepoznavanje mirisa koda. Tako kreirani skupovi podataka se mogu koristiti od strane ML istraživača za obučavanje ML modela za prepoznavanje mirisa koda. Na kraju, socio-ekonomski doprinos se ogleda u mogućnosti upotrebe ML modela za prepoznavanje mirisa obučenih na skupovima podataka koji su rezultat predloženog rešenja. Softverski inženjeri koji teže poboljšanju održivosti koda mogu koristiti ML modele kako bi povećali kvalitet softvera i smanjili ukupne troškove razvoja softvera.

Buduće istraživanje bi trebalo usmeriti na empirijsko potvđivanje zahteva koji definiše da rešenje treba da ubrza anotiranje. Tačnije, trebalo bi usmeriti resurse na formiranje kontrolne grupe anotatora koja bi anotirala mirise bez primene predloženog rešenja ili primenom drugih postojećih rešenja koja se mogu pronaći u literaturi. Nakon toga bi bilo moguće uporediti

vremena anotiranja i proceniti da li je definisani zahtev ispunjen. Time bi se empirijski dokazala i hipoteza kojom se pretpostavlja da rešenje ubrzava anotiranje.

Drugi pravac budućeg istraživanja je validiranje rešenja sa većim brojem anotatora, čime bi se stekao uvid u izazove sa kojima se susreće veća grupa anotatora. Zatim, validiranje rešenja na većem skupu mirisa koda može omogućiti dublju analizu primenjivosti i generalizacije rešenja na sve Fowlerove mirise. Ovome bi moglo doprineti integrisanje analize novih svojstava koda u anotacionu proceduru. Na primer, istorija promena bi dala uvid u razvoj i evoluciju softvera, što može biti značajno za neke Fowlerove mirise koda.

LITERATURA

[1] *IEEE Glossary of software engineering terminology*, IEEE Standard 610.12-1990, 1990.

[2] *ISO/IEC 25010:2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*, International Organization for Standardization, 2011. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

[3] H. Van Vliet, H. Van Vliet and J.C. Van Vliet, *Software engineering: principles and practice*, Vol. 13, Hoboken, NJ: John Wiley & Sons, 2008.

[4] R.S. Pressman, *Software engineering: a practitioner's approach*. Palgrave macmillan, 2005.

[5] M.W. Mkaouer, M. Kessentini, M.Ó. Cinnéide, S. Hayashi and K. Deb, „A robust multi-objective approach to balance severity and importance of refactoring opportunities," *Empirical Software Engineering*, 22, pp. 894-927, 2017. https://doi.org/10.1007/s10664-016-9426-8

[6] G. Vale, R. Abílio, A. Freire and H. Costa, „Criteria and guidelines to improve software maintainability in software product lines," in *2015 12th International Conference on Information Technology-New Generations*, pp. 427-432, IEEE, April 2015. https://doi.org/10.1109/ITNG.2015.75

[7] T.O. Lehtinen, M.V. Mäntylä, J. Vanhanen, J. Itkonen and C. Lassenius, „Perceived causes of software project failures–An analysis of their relationships," *Information and Software Technology*, *56*(6), pp. 623-643, 2014. https://doi.org/10.1016/j.infsof.2014.01.015

[8] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[9] T. Sharma and D. Spinellis, „A survey on software smells," *Journal of Systems and Software*, *138*, pp. 158-173, 2018. https://doi.org/10.1016/j.jss.2017.12.034

[10] A. Kaur, „A systematic literature review on empirical analysis of the relationship between code smells and software quality attributes," *Archives of Computational Methods in Engineering*, *27*, pp. 1267-1296, 2020. https://doi.org/10.1007/s11831-019-09348-6

[11] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 2018.

[12] M.I. Azeem, F. Palomba, L. Shi and Q. Wang, „Machine learning techniques for code smell detection: A systematic literature review and meta-analysis," *Information and Software Technology*, *108*, pp. 115-138, 2019. https://doi.org/10.1016/j.infsof.2018.12.009

[13] F.A. Fontana, M.V. Mäntylä, M. Zanoni and A. Marino, „Comparing and experimenting machine learning techniques for code smell detection," *Empirical Software Engineering, 21*, pp. 1143-1191, 2016. https://doi.org/10.1007/s10664-015-9378-4

[14] B. Bafandeh Mayvan, A. Rasoolzadegan and A. Javan Jafari, „Bad smell detection using quality metrics and refactoring opportunities," *Journal of Software: Evolution and Process*, *32*(8), p.e2255, 2020. https://doi.org/10.1002/smr.2255

[15] A. Kovačević, J. Slivka, D. Vidaković, K.G. Grujić, N. Luburić, S. Prokić and G. Sladić, „Automatic detection of Long Method and God Class code smells through neural source code embeddings," *Expert Systems with Applications*, *204*, p.117607, 2022. https://doi.org/10.1016/j.eswa.2022.117607

[16] F.A. Fontana, J. Dietrich, B. Walter, A. Yamashita and M. Zanoni, „Antipattern and code smell false positives: Preliminary conceptualization and classification," In *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)* (Vol. 1, pp. 609-613). IEEE, March 2016. https://doi.org/10.1109/SANER.2016.84

[17] M. Zakeri-Nasrabadi, S. Parsa, E. Esmaili and F. Palomba, „A systematic literature review on the code smells datasets and validation mechanisms," *ACM Journal on Computing and Cultural Heritage*, 2023. https://doi.org/10.1145/3596908

[18] T. Lewowski and L. Madeyski, „Code smells detection using artificial intelligence techniques: A business-driven systematic review," *Developments in Information & Knowledge Management for Business Applications: Volume 3*, pp. 285-319, 2022. https://doi.org/10.1007/978-3-030-77916-0_12

[19] A. AbuHassan, M. Alshayeb and L. Ghouti, „Software smell detection techniques: A systematic literature review," *Journal of Software: Evolution and Process, 33*(3), p.e2320, 2021. https://doi.org/10.1002/smr.2320

[20] R.S. Menshawy, A.H. Yousef and A. Salem, „Code smells and detection techniques: a survey," In *2021 international mobile, intelligent, and ubiquitous computing conference (MIUCC)* (pp. 78-83). IEEE, May 2021. https://doi.org/10.1109/MIUCC52538.2021.9447669

[21] M. Hozano, A. Garcia, B. Fonseca and E. Costa, „Are you smelling it? Investigating how similar developers detect code smells," *Information and Software Technology*, *93*, pp. 130-146, 2018. https://doi.org/10.1016/j.infsof.2017.09.002

[22] M.V. Mantyla, J. Vanhanen and C. Lassenius, „Bad smells-humans as code critics," In *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.* (pp. 399-408). IEEE, September 2004. https://doi.org/10.1109/ICSM.2004.13578251

[23] D. Taibi, A. Janes and V. Lenarduzzi, „How developers perceive smells in source code: A replicated study," *Information and Software Technology, 92*, pp. 223-235, 2017. https://doi.org/10.1016/j.infsof.2017.08.008

[24] F. Palomba, G. Bavota, M. Di Penta, F. Fasano, R. Oliveto and A. De Lucia, „On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation," In *Proceedings of the 40th International Conference on Software Engineering*, pp. 482-482, May 2018. https://doi.org/10.1007/s10664-017-9535-z

[25] D. Di Nucci, F. Palomba, D.A. Tamburri, A. Serebrenik and A. De Lucia, „Detecting code smells using machine learning techniques: are we there yet?," In *2018 ieee 25th international conference on software analysis, evolution and reengineering (saner)*, pp. 612-621, IEEE, March 2018. https://doi.org/10.1109/SANER.2018.8330266

[26] H. Grodzicka, A. Ziobrowski, Z. Łakomiak, M. Kawa and L. Madeyski, „Code smell prediction employing machine learning meets emerging Java language constructs," in *Data-Centric Business and Applications: Towards Software Development (Volume 4)*, pp. 137-167, 2020. https://doi.org/10.1007/978-3-030-34706-2_8

[27] L. Madeyski and T. Lewowski, „MLCQ: Industry-relevant code smell data set," In *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering*, pp. 342-347, April 2020. https://doi.org/10.1145/3383219.3383264

[28] M. Hozano, N. Antunes, B. Fonseca and E. Costa, „Evaluating the accuracy of machine learning algorithms on detecting code smells for different developers," In *International Conference on Enterprise Information Systems*, Vol. 2, pp. 474-482, April 2017. https://doi.org/10.5220/0006338804740482

[29] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia and D. Poshyvanyk, „Detecting bad smells in source code using change history information," In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 268-278, IEEE, November 2013. https://doi.org/10.1109/ASE.2013.6693086

[30] F. Palomba, D. Di Nucci, M. Tufano, G. Bavota, R. Oliveto, D. Poshyvanyk and A. De Lucia, „Landfill: An open dataset of code smells with public evaluation," In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pp. 482-485, IEEE, May 2015. https://doi.org/10.1109/MSR.2015.69

[31] F. Palomba, G. Bavota, M. Di Penta, F. Fasano, R. Oliveto and A. De Lucia, „A large-scale empirical study on the lifecycle of code smell co-occurrences," *Information and Software Technology*, *99*, pp. 1-10, 2018. https://doi.org/10.1016/j.infsof.2018.02.004

[32] F. Khomh, S. Vaucher, Y.G. Guéhéneuc and H. Sahraoui, „BDTEX: A GQM-based Bayesian approach for the detection of antipatterns," *Journal of Systems and Software*, *84*(4), pp. 559-572, 2011. https://doi.org/10.1016/j.jss.2010.11.921

[33] F. Khomh, S. Vaucher, Y.G. Guéhéneuc and H. Sahraoui, „A bayesian approach for the detection of code and design smells," In *2009 Ninth International Conference on Quality Software*, pp. 305-314, IEEE, August 2009. https://doi.org/10.1109/QSIC.2009.47

[34] S. Bryton, F.B. e Abreu and M. Monteiro, „Reducing subjectivity in code smells detection: Experimenting with the long method," In *2010 Seventh International Conference on the Quality of Information and Communications Technology*, pp. 337-342, IEEE, September 2010. https://doi.org/10.1109/QUATIC.2010.60

[35] S. Vaucher, F. Khomh, N. Moha and Y.G. Guéhéneuc, „Tracking design smells: Lessons from a study of god classes," In *2009 16th working conference on reverse engineering*, pp. 145-154, IEEE, October 2009. https://doi.org/10.1109/WCRE.2009.23

[36] P. Röttger, B. Vidgen, D. Hovy and J.B. Pierrehumbert, „Two contrasting data annotation paradigms for subjective NLP tasks," In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2022. https://doi.org/10.18653/v1/2022.naacl-main.13

[37] J. Pustejovsky and A. Stubbs, *Natural Language Annotation for Machine Learning: A guide to corpus-building for applications*. O'Reilly Media, Inc, 2012.

[38] J.A.M. Santos, J.B. Rocha-Junior and M.G. de Mendonça, „Investigating factors that affect the human perception on god class detection: an analysis based on a family of four controlled experiments," *Journal of Software Engineering Research and Development*, *5*(1), pp. 1-39, 2017. https://doi.org/10.1186/s40411-017-0042-0

[39] Y. Oortwijn, T. Ossenkoppele and A. Betti, „Interrater disagreement resolution: A systematic procedure to reach consensus in annotation tasks," In *Proceedings of the Workshop on Human Evaluation of NLP Systems (HumEval)*, pp. 131-141, April 2021.
[40] A.M. Davani, M. Díaz and V. Prabhakaran, „Dealing with disagreements: Looking beyond the majority vote in subjective annotations," *Transactions of the Association for Computational Linguistics*, *10*, pp. 92-110, 2022. https://doi.org/10.1162/tacl_a_00449
[41] J. Tu, G. Yu, C. Domeniconi, J. Wang, G. Xiao and M. Guo, „Multi-label crowd consensus via joint matrix factorization," *Knowledge and Information Systems*, *62*, pp. 1341-1369, 2020. https://doi.org/10.1007/s10115-019-01386-7
[42] N. Ide and J. Pustejovsky, *Handbook Of Linguistic Annotation*. vol. 1, Berlin: Springer, 2017. https://doi.org/10.1007/978-94-024-0881-2
[43] S.G. Ganesh, T. Sharma and G. Suryanarayana, „Towards a Principle-based Classification of Structural Design Smells," *J. Object Technol., 12*, 1: pp. 1-29, 2013. 10.5381/jot.2013.12.2.a1
[44] G. Suryanarayana, G. Samarthyam and T. Sharma, *Refactoring for Software Design Smells Managing Technical Debt*. 2014.
[45] G. Lacerda, F. Petrillo, M. Pimenta and Y.G. Guéhéneuc, „Code smells and refactoring: A tertiary systematic review of challenges and observations," *Journal of Systems and Software, 167, Article 110610*, 2020. https://doi.org/10.1016/j.jss.2020.110610
[46] M. Mantyla, J. Vanhanen and C. Lassenius, „A taxonomy and an initial empirical study of bad smells in code," In *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings,* pp. 381-384, IEEE, September 2003. https://doi.org/10.1109/ICSM.2003.1235447
[47] W.C. Wake, *Refactoring Workbook*. Addison-Wesley Longman Publishing Co., Inc, 2003.
[48] G. Rasool and Z.A. Arshad, „Lightweight Approach for Detection of Code Smells," in *Arab J Sci Eng 42*, pp. 483–506, 2017. https://doi.org/10.1007/s13369-016-2238-8
[49] W. Li and R. Shatnawi, „An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution," *Journal of Systems and Software*, 80(7): pp. 1120–1128, 2007. https://doi.org/10.1016/j.jss.2006.10.018
[50] R. Shatnawi and L. Wei, „An investigation of Bad Smells in object-oriented design," *Third International Conference on Information Technology: New Generations (ITNG 2006)*, pp. 161–165, 2006. 10.1109/ITNG.2006.31
[51] A. Yamashita and L. Moonen, „Exploring the impact of inter-smell relations on software maintainability: an empirical study," *International Conference on Software Engineering (ICSE)*, IEEE, pp. 682–691, 2013. 10.1109/ICSE.2013.6606614
[52] M. Abbes, F. Khomh, Y.G. Gueheneuc and G. Antoniol, „An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension," *Proceedings of the 2011 15th European Conference on Software Maintenance and Reengineering, CSMR '11*, IEEE Computer Society, pp. 181-190, 2011. 10.1109/CSMR.2011.24
[53] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice: Using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. 2006. https://doi.org/10.1007/3-540-39538-5
[54] H. Liu, J. Jin, Z. Xu, Y. Bu, Y. Zou and L. Zhang, „Deep Learning Based Code Smell Detection," *IEEE Transactions on Software Engineering*, 1–1, 2020. 10.1109/TSE.2019.2936376
[55] S.R. Chidamber and C.F. Kemerer, „A metrics suite for object oriented design," *IEEE Transactions on Software Engineering, vol. 20, no. 6*, pp. 476-493, June 1994. 10.1109/32.295895
[56] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, D. Poshyvanyk and A. De Lucia, „Mining version histories for detecting code smells," *Software Engineering, IEEE Transactions on, vol. 41, no. 5*, pp. 462–489, May 2015. 10.1109/TSE.2014.2372760
[57] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto and A. De Lucia, „Do They Really Smell Bad? A Study on Developers' Perception of Bad Code Smells," *IEEE International Conference on Software Maintenance and Evolution*, 2014. 10.1109/ICSME.2014.32
[58] F. Pecorelli, D. Di Nucci, C. De Roover and A. De Lucia, „A large empirical assessment of the role of data balancing in machine-learning-based code smell detection," *Journal of Systems and Software, Volume 169*, November 2020. https://doi.org/10.1016/j.jss.2020.110693
[59] J. Schumacher, N. Zazworka, F. Shull, C. Seaman and M. Shaw, „Building empirical support for automated code smell detection," In *Proceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement*, pp. 1-10, September 2010. https://doi.org/10.1145/1852786.1852797
[60] C.B. Seaman, „Qualitative methods in empirical studies of software engineering," *IEEE Transactions on software engineering*, *25*(4), pp. 557-572, 1999. https://doi.org/10.1109/32.799955

[61] S. Prokić, N. Luburić, J. Slivka and A. Kovačević, „Identification of Code Properties that Support Code Smell Analysis,“ In *2023 46th MIPRO ICT and Electronics Convention (MIPRO)*, pp. 1664-1669, IEEE, May 2023. https://doi.org/10.23919/MIPRO57284.2023.10159875
[62] M. Hadj-Kacem and N. Bouassida, „Improving the Identification of Code Smells by Combining Structural and Semantic Information,” in *Neural Information Processing: 26th International Conference*, pp. 296-304, 2019. https://doi.org/10.1007/978-3-030-36808-1_32
[63] E. Murphy-Hill, T. Barik and A.P. Black, „Interactive ambient visualizations for soft advice,” *Information Visualization 12(2)*, pp. 107-132, 2013. https://doi.org/10.1177/1473871612469020
[64] R. Oliveira, L. Sousa, R. De Mello, N. Valentim, A. Lopes, T. Conte, A. Garcia, E. Oliveira and C. Lucena, „Collaborative identification of code smells: A multi-case study,” *IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track*, pp. 33-42, 2017. 10.1109/ICSE-SEIP.2017.7
[65] M. S. Haque, J. Carver and T. Atkison, „Causes, impacts, and detection approaches of code smell: a survey,” *Proceedings of the ACMSE 2018 Conference*, pp. 1-8, 2018. https://doi.org/10.1145/3190645.3190697
[66] A. Kaur, S. Jain, S. Goel and G. Dhiman, „A review on machine-learning based code smell detection techniques in object-oriented software system(s),” *Recent Advances in Electrical & Electronic Engineering, 14(3)*, pp. 290-303, 2021. 10.2174/2352096513999200922125839
[67] G. Rasool and Z. Arshad, „A review of code smell mining techniques,” *Journal of Software: Evolution and Process 27(11)*, pp. 867-895, 2015. https://doi.org/10.1002/smr.1737
[68] B. Walter and B. Pietrzak, „Multi-criteria Detection of Bad Smells in Code with UTA Method,” in *Extreme Programming and Agile Processes in Software Engineering: 6th International Conference, Proceedings 6*, pp. 154-161, 2005. https://doi.org/10.1007/11499053_18
[69] N. Sae-Lim, S. Hayashi and M. Saeki, „An Investigative Study on How Developers Filter and Prioritize Code Smell,” *IEICE TRANSACTIONS on Information and Systems, 101(7)*, pp. 1733-1742, 2018. https://doi.org/10.1587/transinf.2017KBP0006
[70] R. Wieringa, „Empirical research methods for technology validation: Scaling up to practice,“ *Journal of systems and software, 95*, pp. 19-31, 2014. https://doi.org/10.1016/j.jss.2013.11.1097
[71] A. Kovačević, N. Luburić, J. Slivka, S. Prokić, K.G. Grujić, D. Vidaković and G. Sladić, „Automatic detection of code smells using metrics and CodeT5 embeddings: a case study in C#,“ in *Neural Comput & Applic 36*, pp. 9203–9220, 2024. https://doi.org/10.1007/s00521-024-09551-y
[72] M. Y. Mhawish and M. Gupta, “Predicting Code Smells and Analysis of Predictions: Using Machine Learning Techniques and Software Metrics,” in *Journal of Computer Science and Technology, 35*, pp. 1428-1445, 2020. https://doi.org/10.1007/s11390-020-0323-7
[73] S. Slinger, „Code smell detection in Eclipse,” *Delft University of Technology*, 2005.
[74] M. Hadj-Kacem and N. Bouassida, „Deep representation learning for code smells detection using variational auto-encoder,” *International Joint Conference on Neural Networks*, pp. 1-8, 2019.
[75] T. Hall, M. Zhang, D. Bowes and Y. Sun, „Some Code Smells Have a Significant but Small Effect on Faults,” *ACM Transactions on Software Engineering and Methodology, 23(4)*, pp. 1-39, 2014. https://doi.org/10.1145/2629648
[76] S. Mekruksavanich, „Design Flaws Detection in Object-Oriented Software with Analytical Learning Method,“ *International Journal of e-Education, e-Business, e-Management and e-Learning, 1(3)*, p. 210, 2011.
[77] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna and L. Bier, „Clone detection using abstract syntax trees,” *Proceedings International Conference on Software Maintenance (Cat. No. 98CB36272)*, pp. 368-377, 1998. 10.1109/ICSM.1998.738528
[78] A. Tahir, A. Yamashita, S. Licorish, J. Dietrich and S. Counsell, „Can you tell me if it smells? A study on how developers discuss code smells and anti-patterns in Stack Overflow,” *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, pp. 68-78. 2018. https://doi.org/10.1145/3210459.3210466
[79] S. Fu and B. Shen, „Code Bad Smell Detection through Evolutionary Data Mining,” *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 1-9, 2015. 10.1109/ESEM.2015.7321194
[80] S.A. Vidal, C. Marcos and J.A. Díaz-Pace, „An approach to prioritize code smells for refactoring,” in *Automated Software Engineering, 23*, pp. 501-532, 2016. https://doi.org/10.1007/s10515-014-0175-x

[81] D. Rapu, S. Ducasse, T. Gîrba and R. Marinescu, „Using history information to improve design flaws detection," *Eighth European Conference on Software Maintenance and Reengineering*, pp. 223-232, 2004. 10.1109/CSMR.2004.1281423

[82] M. V. Mäntylä and C. Lassenius, „Subjective evaluation of software evolvability using code smells: An empirical study," in *Empirical Software Engineering, 11*, pp. 395-431, 2006. https://doi.org/10.1007/s10664-006-9002-8

[83] K. Dhambri, H. Sahraoui and P. Poulin, „Visual Detection of Design Anomalies," *12th European Conference on Software Maintenance and Reengineering*, pp. 279-283, 2008. 10.1109/CSMR.2008.4493326

[84] D. Jiang, P. Ma, X. Su and T. Wang, „Distance metric based divergent change bad smell detection and refactoring scheme analysis," *International Journal of Innovative Computing, Information and Control, 10(4)*, pp. 1519-1531, 2014.

[85] A. Jermakovics, R. Moser, A. Sillitti and G. Succi, „Visualizing software evolution with lagrein," *Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, pp. 749-750, 2008. https://doi.org/10.1145/1449814.1449843

[86] F.A. Fontana, V. Ferme and M. Zanoni, „Towards Assessing Software Architecture Quality by Exploiting Code Smell Relations," *2015 IEEE/ACM 2nd International Workshop on Software Architecture and Metrics*, pp. 1-7, 2015. https://doi.org/10.1109/SAM.2015.8

[87] R. Wettel and M. Lanza, „Visually localizing design problems with disharmony maps," *Proceedings of the 4th ACM Symposium on Software Visualization*, pp. 155-164. 2008. https://doi.org/10.1145/1409720.1409745

[88] F.A. Fontana and M. Zanoni, „Code smell severity classification using machine learning techniques," *Knowledge-Based Systems, 128*, pp. 43-58, 2017. https://doi.org/10.1016/j.knosys.2017.04.014

[89] Y. Wang, S. Hu, L. Yin and X. Zhou, „Using code evolution information to improve the quality of labels in code smell datasets," In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1, pp. 48-53, IEEE, July 2018. https://doi.org/10.1109/COMPSAC.2018.00015

[90] F. Pecorelli, F. Palomba, D. Di Nucci and A. De Lucia, „Comparing heuristic and machine learning approaches for metric-based code smell detection," In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, pp. 93-104, IEEE, May 2019. https://doi.org/10.1109/ICPC.2019.00023

[91] A. Barbez, F. Khomh and Y.G. Guéhéneuc, „A machine-learning based ensemble method for anti-patterns detection," *Journal of Systems and Software*, *161*, p. 110486, 2020. https://doi.org/10.1016/j.jss.2019.110486

[92] A. Maiga, N. Ali, N. Bhattacharya, A. Sabané, Y.G. Guéhéneuc and E. Aimeur, „Smurf: A svm-based incremental anti-pattern detection approach," In *2012 19th Working Conference on Reverse Engineering*, pp. 466-475, IEEE, October 2012. https://doi.org/10.1109/WCRE.2012.56

[93] J. Kreimer, „Adaptive detection of design flaws," *Electronic Notes in Theoretical Computer Science, 141*(4), pp. 117-136, 2005. https://doi.org/10.1016/j.entcs.2005.02.059

[94] J. Yang, K. Hotta, Y. Higo, H. Igaki and S. Kusumoto, „Classification model for code clones based on machine learning," in *Empirical Software Engineering*, *20*, pp. 1095-1125, 2015. https://doi.org/10.1007/s10664-014-9316-x

[95] S. Hassaine, F. Khomh, Y.G. Guéhéneuc and S. Hamel, „IDS: An immune-inspired approach for the detection of software design smells," In *2010 Seventh International Conference on the Quality of Information and Communications Technology*, pp. 343-348, IEEE, September 2010. https://doi.org/10.1109/QUATIC.2010.61

[96] F. Pecorelli, F. Palomba, F. Khomh and A. De Lucia, „Developer-driven code smell prioritization," In *Proceedings of the 17th International Conference on Mining Software Repositories*, pp. 220-231, June 2020. https://doi.org/10.1145/3379597.3387457

[97] M. Lanza, S. Ducasse, H. Gall and M. Pinzger, „CodeCrawler - an information visualization tool for program comprehension," *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, St. Louis, MO, USA, pp. 672-673, 2005. https://doi.org/10.1109/ICSE.2005.1553647

[98] M. Pinzger, H. Gall, M. Fischer and M. Lanza, „Visualizing multiple evolution metrics," In *Proceedings of the 2005 ACM symposium on Software visualization (SoftVis '05)*, Association for Computing Machinery, New York, NY, USA, pp. 67–75, 2005. https://doi.org/10.1145/1056018.1056027

[99] H. Nandani, M. Saad and T. Sharma, „DACOS—A Manually Annotated Dataset of Code Smells," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, Melbourne, Australia, pp. 446-450, 2023. https://doi.org/10.1109/MSR59073.2023.00067

[100] M. Neves and J. Ševa, „An extensive review of tools for manual annotation of documents," *Briefings in Bioinformatics*, Volume 22, Issue 1, pp. 146–163, January 2021. https://doi.org/10.1093/bib/bbz130
[101] J. Slivka, N. Luburić, S. Prokić, K.G. Grujić, A. Kovačević, G. Sladić and D. Vidaković, „Towards a systematic approach to manual annotation of code smells," *Science of Computer Programming*, *230*, p. 102999, 2023. https://doi.org/10.1016/j.scico.2023.102999
[102] S. Prokić, N. Luburić, J. Slivka and A. Kovačević, „Prescriptive procedure for manual code smell annotation," *Science of Computer Programming, 238,* p. 103168, 2024. https://doi.org/10.1016/j.scico.2024.103168
[103] J. Slivka, N. Luburić, D. Vidaković, A. Kovačević, G. Sladić, K.G. Grujić and S. Prokić, „Clean Code and Design Educational Tool – Clean CaDET," Funded by the Science Fund of the Republic of Serbia, Grant No 6521051, AI-Clean CaDET. https://clean-cadet.github.io/, retrieved: 2024-1-3.
[104] M. Škipina, J. Slivka, N. Luburić and A. Kovačević, „Automatic detection of Feature Envy and Data Class code smells using machine learning," *Expert Systems with Applications*, *243*, p.122855, 2023. https://doi.org/10.1016/j.eswa.2023.122855
[105] I. Brdar, J. Vlajkov, J. Slivka, K.G. Grujić and A. Kovačević, „Semi-supervised detection of Long Method and God Class code smells," In *2022 IEEE 20th Jubilee International Symposium on Intelligent Systems and Informatics (SISY)*, pp. 403-408, IEEE, September 2022. https://doi.org/10.1109/SISY56759.2022.10036248
[106] F. Palomba, R. Oliveto and A. De Lucia, „Investigating code smell co-occurrences using association rule learning: A replicated study," In *2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*, pp. 8-13, IEEE, February 2017. 10.1109/MALTESQUE.2017.7882010
[107] P. Piotrowski and L. Madeyski, „Software defect prediction using bad code smells: A systematic literature review," In *Data-Centric Business and Applications*, pp. 77-99, Springer, Cham, 2020. https://doi.org/10.1007/978-3-030-34706-2_5
[108] E. Sobrinho, A. De Lucia, M. de Almeida Maia, „Systematic literature review on bad smells – 5 W's: which, when, what, who, where," *IEEE Transactions on Software Engineering*, Volume: 47, Issue: 1, January 2021. 10.1109/TSE.2018.2880977
[109] E. Tom, A. Aurum and R. Vidgen, „An exploration of technical debt," *Journal of Systems and Software, vol. 86, no. 6*, pp. 1498-1516, 2013. https://doi.org/10.1016/j.jss.2012.12.052
[110] T. Lewowski and L. Madeyski, „Creating evolving project data sets in software engineering," in *Integrating Research and Practice in Software Engineering*, Springer, Cham, pp. 1-14, 2020. https://doi.org/10.1007/978-3-030-26574-8_1
[111] R. C. Martin, *Clean Code: a Handbook of Agile Software Craftsmanship*, Pearson Education, 2009.
[112] G. A. Campbell, „Cognitive complexity: An overview and evaluation," in *Proceedings of the 2018 International Conference on Technical Debt*, 2018. https://doi.org/10.1145/3194164.3194186
[113] R. C. Martin, J. Newkirk and R. S. Koss, *Agile Software Development: Principles, Patterns, and Practices*, vol. 2, Prentice Hall, 2003.
[114] F. Palomba, A. Panichella, A. Zaidman, R. Oliveto and A. De Lucia, „The scent of a smell: An extensive comparison between textual and structural smells," *IEEE Transactions on Software Engineering, vol. 44, no. 10*, pp. 977-1000, 2017. 10.1109/TSE.2017.2752171
[115] A.F. Hayes and K. Krippendorff, K, „Answering the call for a standard reliability measure for coding data," *Communication methods and measures*, *1*(1), pp. 77-89, 2007. https://doi.org/10.1080/19312450709336664
[116] J.R. Landis and G.G. Koch, „The measurement of observer agreement for categorical data," *biometrics*, pp. 159-174, 1977. https://doi.org/10.2307/2529310
[117] A. French, M. Macedo, J. Poulsen, T. Waterson and A. Yu, „Multivariate analysis of variance (MANOVA)," 2008.
[118] S. Seabold and J. Perktold, „Statsmodels: Econometric and statistical modeling with python," in *Proceedings of the 9th Python in Science Conference*, 2010. https://doi.org/10.25080/Majora-92bf1922-011
[119] A. Ng, *Machine learning yearning*. 139, p. 30, 2017. https://github.com/ajaymache/machine-learning-yearning/blob/master/full%20book/machine-learning-yearning.pdf
[120] A. Tahir, J. Dietrich, S. Counsell, S. Licorish and A. Yamashita, „A large scale study on how developers discuss code smells and anti-pattern in stack exchange sites," *Information and Software Technology,* vol. 125, p. 106333, 2020. https://doi.org/10.1016/j.infsof.2020.106333

[121] M. Aniche, „Java code metrics calculator (CK),” [Online] Available: https://github.com/mauricioaniche/ck/ [Accessed 6-3-2025].
[122] L. Madeyski and T. Lewowski, „Detecting code smells using industry-relevant data,“ *Information and Software Technology,* 155, pp. 107112, 2023. https://doi.org/10.1016/j.infsof.2022.107112
[123] T. Lewowski and L. Madeyski, „How far are we from reproducible research on code smell detection? A systematic literature review,“ *Information and Software Technology*, 144, pp. 106783, 2022. https://doi.org/10.1016/j.infsof.2021.106783
[124] F. Caram, B. Rodrigues, A. Campanelli and F. Parreiras, „Machine learning techniques for code smells detection: a systematic mapping study,“ *International Journal of Software Engineering and Knowledge Engineering,* 29, no. 02, pp. 285-316, 2019. https://doi.org/10.1142/S021819401950013X

План третмана података

Назив пројекта/истраживања
Решење за ручно анотирање мириса кода засновано на прескриптивној анотационој парадигми
Назив институције/институција у оквиру којих се спроводи истраживање
Факултет техничких наука, Универзитет у Новом Саду
Назив програма у оквиру ког се реализује истраживање
J. Slivka, N. Luburić, D. Vidaković, A. Kovačević, G. Sladić, K.G. Grujić and S. Prokić, „Clean Code and Design Educational Tool – Clean CaDET,“ Funded by the Science Fund of the Republic of Serbia, Grant No 6521051, AI-Clean CaDET Рачунарство и аутоматика – докторска дисертација
1. Опис података

1.1 Врста студије

Укратко описати тип студије у оквиру које се подаци прикупљају
Студија спроведена у оквиру дисертације је обухватила истраживање са циљем тестирања постављених хипотеза и валидације предложеног решења. Спроведен је експеримент како би се валидирало решење за ручно анотирање мириса кода.

1.2 Врсте података
а) **квантитативни**
б) квалитативни

1.3. Начин прикупљања података
а) **анкете, упитници, тестови**
б) клиничке процене, медицински записи, електронски здравствени записи
в) генотипови: навести врсту _____
г) административни подаци: навести врсту _____
д) узорци ткива: навести врсту_____
ђ) снимци, фотографије: навести врсту_____
е) текст, навести врсту _____
ж) мапа, навести врсту _____
з) остало: описати _____

1.3 Формат података, употребљене скале, количина података

1.3.1 Употребљени софтвер и формат датотеке:
а) Excel фајл, датотека __.xlsx_____
b) SPSS фајл, датотека _____
c) PDF фајл, датотека _____
d) Текст фајл, датотека _____
e) JPG фајл, датотека _____
f) Остало, датотека _____

1.3.2. Број записа (код квантитативних података)

а) број варијабли __5 мириса кода__
б) број мерења (испитаника, процена, снимака и сл.) ___4154 анотираних исечака кода, 3 анотатора__

1.3.3. Поновљена мерења
а) да
б) **не**

Уколико је одговор да, одговорити на следећа питања:
а) временски размак између поновљених мера је _____
б) варијабле које се више пута мере односе се на _____
в) нове верзије фајлова који садрже поновљена мерења су именоване као _____

Напомене: _____

Да ли формати и софтвер омогућавају дељење и дугорочну валидност података?
*а) **Да***
б) Не
Ако је одговор не, образложити _____
_____

2. Прикупљање података

2.1 Методологија за прикупљање/генерисање података
Три анотатора су анотирала пет мириса кода. Укупно је анотирано 4154 исечака кода. За сваки исечак кода су одређивали интензитет мириса кода (0 - нема мириса, 1 - постоји мирис који није критичан, 2 - постоји мирис који би требало уклонити, 3 - постоји критичан мирис кода).
2.1.1. У оквиру ког истраживачког нацрта су подаци прикупљени?
а) **експеримент**, навести тип __У експерименту је валидирано предложено решење за ручно анотирање мириса кода. Експеримент је спроведен између 2021. и 2023. године. _____
б) корелационо истраживање, навести тип _____
ц) анализа текста, навести тип _____
д) остало, навести шта _____

2.1.2 Навести врсте мерних инструмената или стандарде података специфичних за одређену научну дисциплину (ако постоје).
____Коришћен је *DataSet Explorer* алат, који је дизајниран и имплементиран такође у склопу овог истраживања. Алат је доступан на
https://github.com/Clean-CaDET/dataset-explorer
https://github.com/Clean-CaDET/platform-explorer-ui-web
https://github.com/Clean-CaDET/dataset-explorer/wiki_____

2.2 Квалитет података и стандарди

2.2.1. Третман недостајућих података
а) Да ли матрица садржи недостајуће податке? Да **Не**

Ако је одговор да, одговорити на следећа питања:
а) Колики је број недостајућих података? _____
б) Да ли се кориснику матрице препоручује замена недостајућих података? Да Не
в) Ако је одговор да, навести сугестије за третман замене недостајућих података
_____

2.2.2. На који начин је контролисан квалитет података? Описати

___У експерименту су учествовала три анотатора. Сваки анотирани исечак кода су анотирала бар два анотатора. У случају конфликта је и трећи анотатор анотирао исти исечак кода. Сви конфликти су продискутовани и конфликти су разрешени._____

2.2.3. На који начин је извршена контрола уноса података у матрицу?
_____Алат који је коришћен током експеримента (*DataSet Explorer*) аутоматски извози забележене податке (анотиране исечке кода). Алат осигурава унос само дозвољених вредности (интезитета 0-3) и проверава да ли постоје исечци кода који нису анотирани или анотирани од стране само једног анотатора._____

3. Третман података и пратећа документација

3.1. Третман и чување података

3.1.1. Подаци ће бити депоновани у Репозиторијум докторски дисертација Универзитета у Новом Саду.
3.1.2. URL адреса ___*https://www.cris.uns.ac.rs/etheses.jsf* _____
3.1.3. DOI _____

3.1.4. Да ли ће подаци бити у отвореном приступу?
а) ***Да.***
Анотирани скуп података је доступан и на:
https://zenodo.org/records/6520056#.Y2O1wnbMKUk
https://doi.org/10.5281/zenodo.10475432

б) Да, али после ембарга који ће трајати до _____
в) Не

Ако је одговор не, навести разлог _____

3.1.5. Подаци неће бити депоновани у репозиторијум, али ће бити чувани.
Образложење
_____

_____

3.2 Метаподаци и документација података
3.2.1. Који стандард за метаподатке ће бити примењен? __Стандард који примењује Репозиторијум Универзитета у Новом Саду. __

3.2.1. Навести метаподатке на основу којих су подаци депоновани у репозиторијум.
_____Симона Прокић (2025): Решење за ручно анотирање мириса кода засновано на прескриптивној анотационој парадигми _____

_____

Ако је потребно, навести методе које се користе за преузимање података, аналитичке и процедуралне информације, њихово кодирање, детаљне описе варијабли, записа итд.
_____

_____

_____

_____

3.3 Стратегија и стандарди за чување података
3.3.1. До ког периода ће подаци бити чувани у репозиторијуму? _____
3.3.2. Да ли ће подаци бити депоновани под шифром? Да **Не**
3.3.3. Да ли ће шифра бити доступна одређеном кругу истраживача? Да **Не**
3.3.4. Да ли се подаци морају уклонити из отвореног приступа после извесног времена?
Да **Не**
Образложити
_____

4. Безбедност података и заштита поверљивих информација

Овај одељак МОРА бити попуњен ако ваши подаци укључују личне податке који се односе на учеснике у истраживању. За друга истраживања треба такође размотрити заштиту и сигурност података.
4.1 Формални стандарди за сигурност информација/података
Истраживачи који спроводе испитивања с људима морају да се придржавају Закона о заштити података о личности *(https://www.paragraf.rs/propisi/zakon_o_zastiti_podataka_o_licnosti.html)* и одговарајућег институционалног кодекса о академском интегритету.

4.1.2. Да ли је истраживање одобрено од стране етичке комисије? Да **Не**
Ако је одговор Да, навести датум и назив етичке комисије која је одобрила истраживање
_____

4.1.2. Да ли подаци укључују личне податке учесника у истраживању? Да **Не**
Ако је одговор да, наведите на који начин сте осигурали поверљивост и сигурност информација везаних за испитанике:
а) Подаци нису у отвореном приступу
б) Подаци су анонимизирани
ц) Остало, навести шта
_____
_____

5. Доступност података

5.1. Подаци ће бити
*а) **јавно доступни***
б) доступни само уском кругу истраживача у одређеној научној области
ц) затворени

Ако су подаци доступни само уском кругу истраживача, навести под којим условима могу да их користе:
_____

_____

Ако су подаци доступни само уском кругу истраживача, навести на који начин могу приступити подацима: _____

_____

5.4. Навести лиценцу под којом ће прикупљени подаци бити архивирани.
_____ Ауторство – некомерцијално – без прераде _____

6. Улоге и одговорност

6.1. Навести име и презиме и мејл адресу власника (аутора) података

_____Симона Прокић, simona.prokic@uns.ac.rs_____

6.2. Навести име и презиме и мејл адресу особе која одржава матрицу с подацима

_____Симона Прокић, simona.prokic@uns.ac.rs_____

6.3. Навести име и презиме и мејл адресу особе која омогућује приступ подацима другим истраживачима

_____Симона Прокић, simona.prokic@uns.ac.rs_____