УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

МОДЕЛ ОЦЕЊИВАЊА СОФТВЕРСКИХ ПРОЈЕКАТА ЗАСНОВАН НА СТАТИЧКОЈ АНАЛИЗИ КÔДА

ДОКТОРСКА ДИСЕРТАЦИЈА

Ментор:
проф. др Дарко Стефановић

Кандидат:
Данило Николић

Нови Сад, 2024. године

УНИВЕРЗИТЕТ У НОВОМ САДУ **ОБРАЗАЦ – 5а**
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА[1]

Врста рада:	Докторска дисертација
Име и презиме аутора:	Данило Николић
Ментор (титула, име, презиме, звање, институција)	др Дарко Стефановић, редовни професор, Факултет техничких наука, Универзитет у Новом Саду
Наслов рада:	Модел оцењивања софтверских пројеката заснован на статичкој анализи кôда
Језик и писмо рада:	Српски, ћирилица
Физички опис рада:	Унети број: Страница 132 Поглавља 7 Референци 107 Табела 27 Слика 30 Графикона 0 Прилога 4
Научна област:	Информационе технологије: Индустријско инжењерство и инжењерски менаџмент; и Електротехничко и рачунарско инжењерство (образовно-научно поље: Техничко-технолошке науке)
Ужа научна област (научна дисциплина):	Инжењерство информационих система
Кључне речи / предметна одредница:	Квалитет кôда, статичка анализа кôда, едукација, оцењивање пројеката, плагијаризам
Апстракт на језику рада:	Фокус ове докторске дисертације је анализа фактора који утичу на квалитет изворног кôда софтверских пројеката и корелацију између квалитета кôда и академског успеха. За идентификацију одредница коришћена је статичка анализа кôда као основни алат. Осим тога, дисертација обухвата истраживање проблема плагијаризма у изворном кôду и потребу за имплементацијом алата за аутоматско оцењивање пројеката на бази статичке анализе. На основу анализираних хипотеза и модела истраживања, утврђени су кључни фактори који представљају одреднице квалитета кôда, корелација квалитета и академског успеха, и дате су смернице за интеграцију статичке анализе кôда у процес оцењивања софтверских пројеката.
Датум прихватања теме од стране надлежног већа:	24.10.2024.
Датум одбране: (Попуњава накнадно институција)	
Чланови комисије: (титула, име, презиме,звање, институција)	Председник: др Соња Ристић, редовни професор, Универзитет у Новом Саду Факултет техничких наука Члан: др Срђан Попов, редовни професор, Универзитет у Новом Саду, Факултет техничких наука Члан: др Душан Савић, ванредни професор, Универзитет у Београду, Факултет организационихнаука Члан: др Мирослав Стефановић, доцент, Универзитет у Новом Саду, Факултет техничких наука Члан: др Душанка Дакић, доцент, Универзитет у Новом Саду Факултет техничких наука Ментор: др Дарко Стефановић, редовни професор, Универзитет у Новом Саду Факултет техничких наука
Напомена:	

[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.

UNIVERSITY OF NOVI SAD

FACULTY OF TECHNICAL SCIENCES

KEY WORD DOCUMENTATION[2]

Document type:	Doctoral dissertation
Author:	Danilo Nikolić
Supervisor (title, first name, last name, position, institution)	dr Darko Stefanovic, full professor, University of Novi Sad Faculty of Technical Sciences
Thesis title:	Model for Evaluating Software Projects Based on Static Code Analysis
Language of text(script):	Serbian language (cyrillic script)
Physical description:	Number of:Pages 132 Chapters 7 References 107 Tables 27 Illustrations 30 Graphs 0 Appendices 4
Scientific field:	Information Technologies: Industrial Engineering and Engineering Management; and Electrical and Computer Engineering (educational-scientific field: Technical-Technological Sciences)
Scientific subfield (scientific discipline):	Information Systems Engineering
Subject, Keywords:	Code quality, static code analysis, education, project evaluation, plagiarism
Abstract in English language:	The focus of this doctoral dissertation is the analysis of factors that influence the quality of source code in software projects and the correlation between code quality and academic success. Static code analysis was employed as the primary tool for identifying these factors. Additionally, the dissertation includes research on the issue of plagiarism in source code and the need for implementing tools for automatic project assessment based on static analysis. Based on the analyzed hypotheses and research models, the key factors influencing code quality and the correlation between quality and academic success were identified, and guidelines for integrating static code analysis into the process of evaluating software projects were provided.
Date of endorsment by the scientific board:	24.10.2024.
Defended: (Filled by the faculty service)	
Thesis Defend Board: (title, first name, last name, position, institution)	President: Dr. Sonja Ristić, full professor, University of Novi Sad, Faculty of Technical Sciences Member: Dr. Srđan Popov, full professor, University of Novi Sad, Faculty of Technical Sciences Member: Dr. Dušan Savić, associate professor, University of Belgrade, Faculty of Organizational Sciences Member: Dr. Miroslav Stefanović, assistant professor, University of Novi Sad, Faculty of Technical Sciences Member: Dr. Dušanka Dakić, assistant professor, University of Novi Sad, Faculty of Technical Sciences Mentor: Dr. Darko Stefanović, full professor, University of Novi Sad, Faculty of Technical Sciences
Note:	

[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.

Захвалност

Желим да се захвалим ментору, проф. др Дарку Стефановићу на искреној помоћи и подршци у научном раду и приликом писања дисертације.

Такође, захваљујем се колегама са Катедре за информационо-комуникационе системе, који су својим сугестијама пружили подршку приликом развоја идејног концепта и спровођења истраживања.

Посебну захвалност дугујем родитељима Слободанки Николић и Зорану Николићу и брату Милошу Николићу.

На мој лични и професионални развој, који је у крајњем исходу довео до настанка ове докторске дисертације, утицали су и следећи људи, при чему редослед њиховог навођења није од значаја:

Сара Копривица, др Душанка Дакић, др Мирослав Стефановић, Мирослав Николић, Сара Кијановић, Алекса Комосар, Марко Штака, Александра Хорњак, Анђела Тодорић, Душан Крстић, др Теодора Вучковић, Александар Копривица, Ивана Пејановић, Јована Копривица, Вања Живановић, Марина Барјактаровић, Ана Чолока, Александар Бранков, Михаило Копривица, Стефан Копривица, Јована Маргитин, Небојша Балан, Ђурђина Грујић, Маја Славуљ, Јасмина Тадић, Љубица Николић, Ивана Благојевић, Предраг Пејовић, Зорана Јелицки, Љубомир Симин, Биљана Ракић, др Соња Ристић, др Ђорђе Пржуљ, др Срђан Сладојевић, др Марко Арсеновић, Дајана Антанасијевић, Милош Милашиновић, Дијана Бурсаћ, Наташа Мирков, Владимир Мирков, Урош Мирков, Душан Дугајлић, Предраг Ђуричић, Лазар Пиљојчић, Ана Пиљојчић, Аријана Фишић, Андриана Матанов, Александар Матанов, Јелена Матанов, Љубинка Николић, Мирко Николић, Мирко Николић, Сања Николић, Божана Марков, Недељка Манов, Божидар Манов, Марија Манов, Србинко Марков, др Борис Милашиновић, др Игор Мектеровић, Светлана Баковић, Олгица Стефановић, Весна Мицевска-Антал, Јадранка Огризовић, Данијела Лалић, Тијана Моцељ, Андреј Катин, Зорица Бабић, Александра Остојић, Ненад Милановић, Andrew Garfield, Timothee Chalamet, Галеб Никачевић, Милена Гилезан, Ивана Копривица, Ивана Ружичка, Ellen DeGeneres, Хана Павелка-Стојшић, Вања Попов, Маријана Пејановић, Јелица Ђорић, Урош Тодорић, Велибор Митровић, Михаило Вукобратовић, Лука Вулановић, Радица Виријевић, Драгана Маргитин, Мирко Рајковић, Ивица Маргитин, Наташа Чолока, Денуц Чолока, Далиборка Балан, Ивица Балан, Оља Кузмановић – Куруц, Јана Малушић, Александра Копривица, Нела Благоје, Игор Калин, Анђела Милићевић, Оливера Војциезек, Нина Тасевски, Филип Несторов, Данијел Радаковић, Марина Дражић, Душан Гојков, Сара Алексић, Анита Варга, Атила Варга, Олга Дега, др Ксенија Дорословачки, др Данијела Грачанин, др Срђан Попов, др Душан Савић, др Тамара Продановић, др Барбара Киченхам, Миодраг Мајић,, Вељко Николић, Зорица Николић, Милош Николић, Љубисав Николић, Биљана Цвијовић, Ђорђе Николић, Hans Zimmer…

Садржај

1. УВОДНА РАЗМАТРАЊА 1

1.1 Ток истраживања докторске дисертације 2

1.2 Очекивани резултати истраживања и научни допринос дисертације 5

1.3 Структура дисертације 6

2. ТЕОРИЈСКЕ ОСНОВЕ 8

2.1 Основни концепти статичке анализе кôда 8
2.1.1. Дефиниција и значај квалитета кôда 8
2.1.2. Статичка анализа кôда 9
2.1.3. Технике и методе статичке анализе кôда 10
2.1.4. Алати за статичку анализу кôда 11
2.1.5. Стандард квалитета софтверског производа – *ISO/IEC* 25010:2023 12

2.2 Класификација софтверских пројеката 13

2.3 Преглед стања у области 14
2.3.1. Преглед употребе алата за статичку анализу кôда 14
2.3.2. Преглед примене алата за статичку анализу кôда и одредница квалитета кôда 19
2.3.3. Преглед примене статичке анализе кôда у детекцији плагијаризма и оцењивању 23
2.3.4. Истраживање примене статичке анализе кôда у индустрији 26

2.4 Истраживачка питања, истраживачки модели и хипотезе 27

3. МЕТОДОЛОШКИ АСПЕКТИ ИСТРАЖИВАЊА 33

3.1 Методологија обраде података истраживачког модела 1 (ИМ1) 33
3.1.1. Прикупљање података 33
3.1.2. Обрада података 34
3.1.3. Креирање нове променљиве – квалитет кôда (КК) 42
3.1.4. Избор статистичких тестова и анализа 43
3.1.5. Избор алата за статичку анализу кôда који ће бити коришћен у истраживању 44

3.2 Методологија обраде података истраживачког модела 2 (ИМ2) 52
3.2.1. Развој и дистрибуција мерног иструмента 52
3.2.2. Демографске карактеристике испитаника 54
3.2.3. Обрада података 55
3.2.4. Избор статистичких тестова и метода 59

4. РЕЗУЛТАТИ ИСТРАЖИВАЊА 61

4.1 Резултати истраживачког модела 1 61
4.1.1. Резултати анализе утицаја фактора поставке пројекта на квалитет кôда 61
4.1.2. Анализа корелације између квалитета кôда и академске успешности 65
4.1.3. Валидација резултата 66
4.1.4. Процена и предвиђање квалитета кôда пројеката на општем узроку 69

4.2 Резултати истраживачког модела 2 71

5. ДИСКУСИЈА РЕЗУЛТАТА ИСТРАЖИВАЊА 75

5.1 Дискусија резултата истраживачког модела 1 75
5.1.1. Дискусија резултата испитивања хипотеза ИМ1 75
5.1.2. Дискусија валидације резултата 83
5.1.3. Смернице за организацију и поставку софтверских пројеката 85

5.2 Дискусија резултата истраживачког модела 2 87
5.2.1. Дискусија резултата испитивања хипотезе ИМ2 87
5.2.2. Смернице за организацију процеса оцењивања софтверских пројеката 89

5.3 Ограничење примене истраживачких модела 91

6. ЗАКЉУЧНА РАЗМАТРАЊА И ПРАВЦИ ДАЉЕГ ИСТРАЖИВАЊА 93

6.1 Правци даљег истраживања 94

7. ЛИТЕРАТУРА 96

Прилог А - Опсервације ИМ1 103

Прилог Б – Упитник ИМ2 110

Прилог В - Статистичка анализа (ИМ1) 112

Прилог Г – Опсервације за валидацију резултата 124

Сажетак

Квалитет изворног кôда је кључан елемент успешности било ког софтверског пројекта. У свету савременог софтверског инжењерства који се непрестано мења и где пројекти постају све сложенији, неопходно је стално радити на унапређењу и одржавању високог квалитета кôда. Висок квалитет кôда осигурава ефикасност, робусност и одрживост софтверских производа, што је од суштинске важности за успех на конкурентном технолошком тржишту.

Статичка анализа кôда је истакнута као један од основних метода за одржавање квалитета кôда. Овај метод омогућава идентификацију грешака, сигурносних пропуста и нежељених шаблона у кôду без потребе за његовим извршавањем, чиме се проблеми могу открити пре тестирања или продукционисања. Статичка анализа служи и као превентивна мера која може значајно смањити трошкове развоја софтвера.

У склопу истраживања у дисертацији, разматрано је у којој мери статичка анализа кôда налази примену у наставном процесу и да ли њена употреба у образовном окружењу може бити подједнако корисна као у индустрији. Такође, истражена је могућност интеграције алата за статичку анализу у наставне програме и могућност унапређења студентског разумевања проблема кроз унапређење квалитета кôда.

Истраживање употребе статичке анализе кôда у студентским пројектима је кључно за откривање фактора који утичу на квалитет кôда. У дисертацији су анализирани различити аспекти пројектних поставки, укључујући технолошки избор, тимску сарадњу и методологију наставе, како би се унапредио образовни процес у области софтверског инжењерства и другим областима у којима се развијају софтверска решења.

Потенцијал статичке анализе кôда у образовању такође укључује детекцију плагијаризма и развој метода за аутоматско оцењивање. Истраживање употребе различитих алата и техника за ефикасну аутоматизацију детекције плагијаризма и развој система за оцењивање може олакшати процес оцењивања и унапредити објективност у оцењивању студентских радова.

Summary

Code quality is a key element in the success of any software project. In the constantly changing environment of modern software engineering, where projects become increasingly complex, it is necessary to continually improve and maintain high code quality. High code quality ensures the efficiency, robustness, and sustainability of software products, which is essential for success in the competitive technology market.

Static code analysis stands out as a fundamental method for maintaining code quality. This method allows for the detection of errors, security flaws, and unwanted patterns in code without the need for execution, helping to identify problems before testing or production. Static analysis also serves as a preventative measure that can significantly reduce software development costs.

As part of the research conducted in this dissertation, the extent to which static code analysis is applied in the educational process was examined, along with whether its use in academic environments can be as beneficial as it is in the industry. Additionally, the possibility of integrating static analysis tools into curricula and improving students' understanding of issues through enhancing code quality was explored.

The investigation of the use of static code analysis in student projects is essential for identifying factors that influence code quality. The dissertation analyzes various aspects of project setups, including technological choices, team collaboration, and teaching methodologies, with the goal of improving the educational process in the field of software engineering.

The potential of static code analysis in education also includes plagiarism detection and the development of automatic evaluation methods. Researching the use of various tools and techniques for efficiently automating plagiarism detection and developing evaluation systems can facilitate the evaluation process and improve fairness and objectivity in assessing student work.

Листа слика

Слика 1 – Позиција статичке анализе кôда у поступку развоја софтвера 2
Слика 2 – Ток истраживања докторске дисертације 4
Слика 3 – Генеричке неправилности и неправилности специфичне за контекст 10
Слика 4 – Ток спровођења прегледа стања у области 15
Слика 5 – Расподела примарних студија који описују алате са подршком за језике опште намене 16
Слика 6 – Алати који су највише заступљени у примарним студијама 17
Слика 7 – Ток спровођења другог прегледа стања у области 20
Слика 8 – Ток спровођења трећег прегледа стања у области 24
Слика 9 – Истраживачки модел 1 (ИМ1) 28
Слика 10 – Истраживачки модел 2 (ИМ2) 31
Слика 11 – Расподела броја појављивања метрике багова (грешака) у подацима 37
Слика 12 – Расподела броја појављивања метрике *code smells* у подацима 37
Слика 13 – Расподела броја појављивања метрике duplikacije (%) у подацима 38
Слика 14 – Расподела броја појављивања метрике величина пројекта у подацима 38
Слика 15 – Расподела броја појављивања метрике сигурносних тачки у подацима 39
Слика 16 – Расподела броја појављивања метрике академске године у подацима 40
Слика 17 – Расподела броја појављивања метрике оцена у подацима 40
Слика 18 – Расподела броја појављивања метрике времена потребног за полагање пројекта у подацима 41
Слика 19 – Расподела броја појављивања метрике рањивости у подацима 41
Слика 20 – Расподела броја појављивања метрике употребе алата за проверу плагијаризма у процесу оцењивања софтверских пројеката (ОУАПП) 57
Слика 21 – Расподела броја појављивања метрике оцене у којој мери наставници подстичу студенте да користе алате за статичку анализу кôда у току развоја софтверских решења (ОУСАК) 57
Слика 22 – Расподела броја појављивања метрике оцене важности проблема плагијарзима у области софтверског инжењерства (ПП) 58
Слика 23 – Расподела броја појављивања метрике оцене изазовности прегледа великог броја софтверских пројеката без употребе алата (ИПБА) 58
Слика 24 – Расподела броја појављивања метрике оцене спремности наставника за коришћење алата за процену квалитета кôда и детекцију плагијата (СНКА) 59
Слика 25 – Расподела броја појављивања метрике оцене валидности и адекватности алата за проверу плагијаризма и оцену квалитета кôда у процесу оцењивања (АМП) 59
Слика 26 – *Variance Inflaction Factor (VIF)* за независне варијабле поставке пројекта 62

Слика 27 – Синергетски ефекат величине пројекта и избора технологије на квалитет кôда 65
Слика 28 – Измењен истраживачки модел 1 са циљем анализе података за валидацију 66
Слика 29 – Функција расподеле вероватноћа 70
Слика 30 – Кумулативна дистрибутивна функција 71

Листа табела

Табела 1 – Расподела примарних студија према типу језика за које алати обезбеђују подршку 16
Табела 2 – Примарне студије класификоване према типу неправилности које представљени алати детектују 18
Табела 3 – Синтеза резултата другог прегледа стања у области 22
Табела 4– Синтеза примарних студија трећег прегледа стања у области према фокусу истраживања 25
Табела 5 – Категоризација примарних студија трећег прегледа стања у области према циљној групи на коју се односи 26
Табела 6 – Просечне оцене алата за статичку анализу кôда по искуственим групама софтвер инжењера 27
Табела 7 – Скраћени називи метрика из ИМ1 35
Табела 8– Приказ могућих вредности сваке варијабле 35
Табела 9 – Сетови карактеристика коришћени у анализи 47
Табела 10 – Резултати анализе алата *Cppcheck* 49
Табела 11 – Резултати анализе алата *FindBugs* 50
Табела 12 – Резултати анализе алата *SonarQube* 51
Табела 13 – Преглед резултата свих алата по сетовима карактеристика 52
Табела 14 – Скраћени називи метрика из ИМ2 56
Табела 15 – Корелација између поставке пројекта и квалитета кôда 62
Табела 16 – Резултати биномне логистичке регресије 63
Табела 17 – Корелација између независних варијабли и КК 64
Табела 18 – Резултати корелације између квалитета кôда и академског успеха студената 66
Табела 19 – Дескриптивна статистика података из истраживања за валидацију резултата 68
Табела 20 – Резултати биномне логистичке регресије за хипотезе 1 – 4 69
Табела 21 – Испитивање корелације између квалитета кôда и оцене студента 69
Табела 22 – Резултати корелације између ПП и СНКА 71
Табела 23 – Резултати корелације између ИПБА и СНКА 72
Табела 24 – Број испитаника који користе алате за ПП 73
Табела 25 – Резултати тестирања хипотезе Х12 73
Табела 26 – Резултати тестирања хипотезе Х13 73
Табела 27 – Удео појављивања вредности бинарних варијабли 76

Листа скраћеница

Српски језик

ИТ	**–**	избор технологије
ТС	**–**	тип сарадње
ВП	**–**	величина пројекта
АГ	**–**	академска година
УАВ	**–**	употреба алата за верзионисање
УСАК	**–**	употреба алата за статичку анализу кôда
НОН	**–**	начин одржавања наставе
КК	**–**	квалитет кôда
ОС	**–**	оцена студента
БРП	**–**	број рокова потребних за полагање пројекта
ОУСАК	–	обим употребе алата за статичку анализу кôда
ПП	–	проблем плагијаризма
ОУАПП	–	обим употребе алата за проверу плагијаризма
ИПБА	**–**	изазовност прегледа без употребе алата
СНКА	**–**	спремност за коришћење алата
АМП	**–**	адекватност метода процене
ИМ1	–	истраживачки модел 1
ИМ2	–	истраживачки модел 2

Енглески језик

DESMET	**–**	*Methodology for Evaluating Software Engineering Methods/Tools*
ICET	**–**	*International Conference on Engineering and Technology*
DAAAM	**–**	*Danube Adria Association for Automation and Manufacturing*
IP	**–**	*Internet Protocol*
MOOC	–	*Massive Open Online Courses*
ISO	**–**	*International Organization for Standardization*
IEC	**–**	*Internation Electrotechnical Commision*
VIF	**–**	*Variance Inflation Factor*
CVE	–	*Common Vulnerabilities and Exposures*
IDE	–	*Integrated Development Environment*
SQuaRE	**–**	*Systems and Software Quality Requirements and Evaluation*
PLC	–	*Programmable Logic Controller*
CDF	–	*Cumulative Distribution Function*

1. Уводна разматрања

Висок квалитет изворног кôда (енгл. *source code*) кључан је за успех софтверских пројеката у данашњем динамичном технолошком окружењу [1]. Постизање и одржавање високог нивоа квалитета кôда представља изазов због бројних фактора који утичу на развој пројеката у области софтверског инжењерства. Међу овим факторима, организација и структура пројекта играју значајну улогу у одређивању квалитета изворног кôда. У академском контексту, организација и поставка студентских пројеката варирају у зависности од курса, где се могу наћи тимски или индивидуални пројекти са различитим захтевима за технологије и алате. Ове варијације директно утичу на квалитет кôда који студенти генеришу и пружају основу за истраживање утицаја различитих метода организације на квалитет изворног кôда [2], [3], [4].

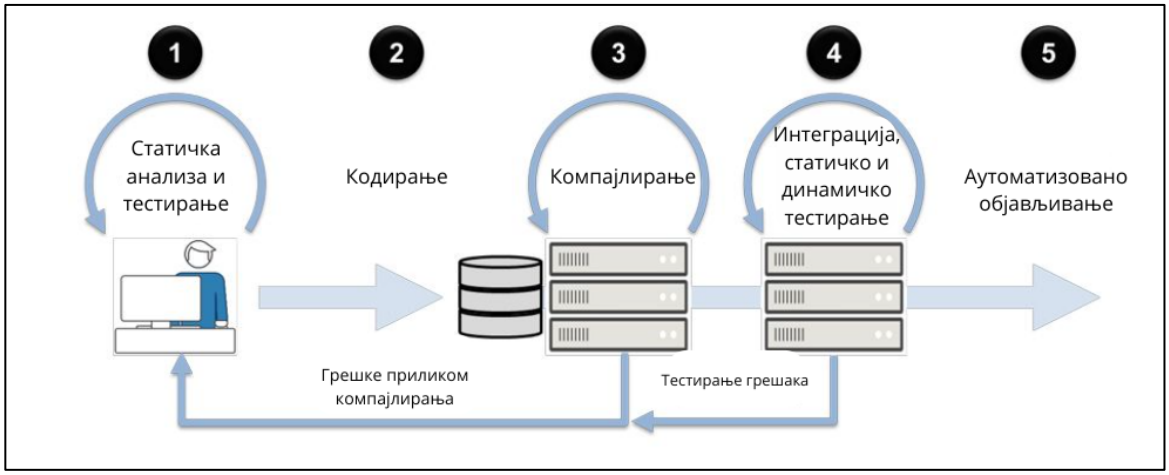
Статичка анализа кôда знатно доприноси процесима оцењивања и управљања квалитетом кôда у образовању будућих инжењера софтвера [5], [6], [7]. Проблеми попут плагијаризма, који укључују копирање пројеката или коришћење изворног кôда генерисаног од стране алата базираних на вештачкој интелигенцији, представљају значајан изазов за наставнике [8], [9]. Ефикасна идентификација и превенција плагијаризма кроз методе статичке анализе може значајно помоћи у очувању академске честитости и квалитета наставног процеса.

С обзиром на велики број студената и различитост пројеката, традиционалне методе прегледа кôда постају временски захтевне и често подложне субјективним грешкама. Аутоматизација оцењивања, која укључује развој алгоритама који могу да процене квалитет кôда базиран на унапред дефинисаним критеријумима, нуди решење за ове изазове [10], [11]. Увођење аутоматизованих система за оцењивање омогућава наставницима да брже и прецизније оцењују радове, што доприноси објективности и праведности у процесу оцењивања.

Ипак, одржавање и континуирано праћење квалитета кôда кључно је како за академске (студентске), тако и за све остале типове пројеката. Како би се могло утицати на одреднице квалитета кода, важно је, пре свега, идентификовати факторе поставке пројекта који могу утицати на квалитет кода, а онда и поставити смернице за организацију и поставку пројекта на начин који омогућава развој квалитетних софтверских решења.

Статичка анализа кôда, заједно са тестирањем, представља континуиран и интегралан део процеса развоја софтверских производа. Након развоја одређене целине софтверског пројекта, кôд пролази кроз фазу анализе и тестирања, како би се идентификовале и исправиле грешке пре његовог пуштања у продукционо окружење. На слици 1 графички је приказана позиција статичке анализе у оквиру процеса развоја софтверских производа.

Поред тога, развој и примена метода статичке анализе кôда у образовном контексту може омогућити студентима да боље разумеју значај и импликације квалитета кôда у својим будућим професионалним ангажманима. Кроз практичну примену алата, студенти могу стећи вештине неопходне за развој висококвалитетних софтверских решења, који су основа за успех на конкурентном технолошком тржишту.



Слика 1 – Позиција статичке анализе кôда у поступку развоја софтвера [1]

1.1 Ток истраживања докторске дисертације

Прва фаза карактерише преглед стања у области који укључује алате за статичку анализу кôда, њихову примену у различитим индустријама, истраживањима и образовању, као и детекцију неисправности кôда. Ова фаза водила је до публикације рада на међународној конференцији која се бави систематским прегледом алата, идентификацијом подршке за различите програмске језике, и увидом у могућности и ограничења алата за језике наменске за домен (енгл. *domain-specific languages*). Публикација са насловом "*Static Code Analysis Tools: A Systematic Literature Review"* [12] објављена је на међународној конференцији 2020. године.

У другој фази, настављено је истраживање путем детаљне анализе и евалуације најкоришћенијих алата идентификованих у првој фази, коришћењем *A Methodology for Evaluating Software Engineering Methods and Tools – DESMET* методологије. Ово води до идентификације *SonarQube*-а као алата са највишом оценом, који је потом коришћен за даља истраживања. Резултати ове анализе алата објављени су на конференцији *Infoteh – Jahorina,* 2021. године, публикација је носила наслов "*Analysis of The Tools for Static Code Analysis*" [13].

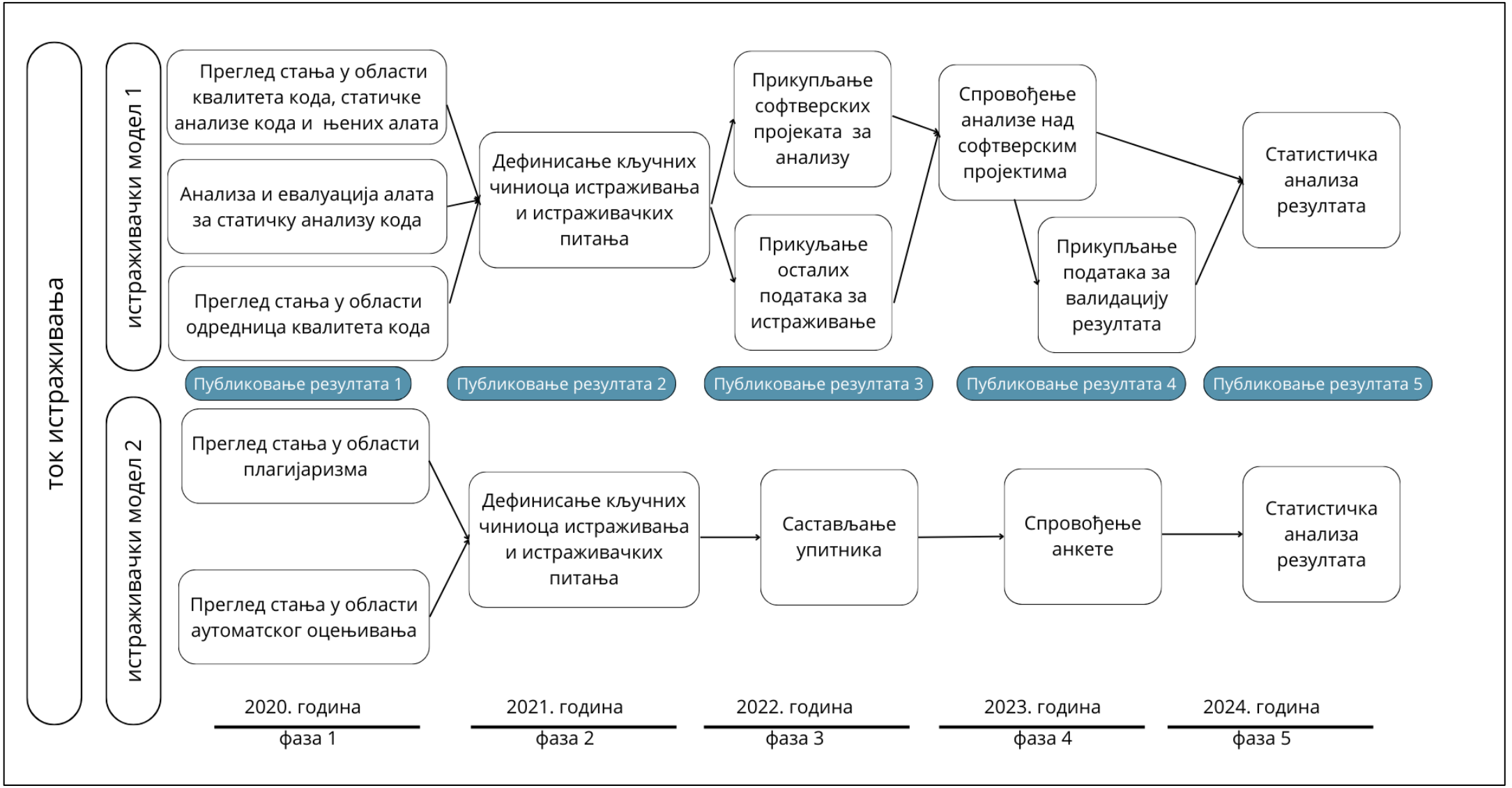
Наставак истраживања у области водио је ка идентфикацији, примени и анализи стратегија за примену статичке анализе кôда. Резултати ове евалуације, као и анализа стратегија примене алата, публиковани су на међународној конференцији *International Conference on Engineering and Technology – ICET*-2021 под насловом „*Identification of strategies over tools for static code analysis*" [14] и на међународној конференцији *Danube Adria Association for Automation and Manufacturing – DAAAM* 2021 под насловом „*Evaluation of Strategies over Static Code Analysis Tools*" [15].

Трећа фаза фокусира се на емпиријска истраживања употребе алата за статичку анализу кôда у индустрији. Истраживања су укључивала анализу учесталости и начина употребе алата међу инжењерима софтвера и њихов утицај на квалитет кôда. Резултати су представљени на међународној конференцији 2023. године под насловом „*On the application of static code analysis tools in the Serbian IT industry: an empirical study*" [16].

Четврта фаза укључује истраживање које је анализирало како различити параметри пројекта утичу на квалитет кôда. Ова анализа обухватала је величину пројекта, употребу алата за статичку анализу кôда и избор технологија и њихову корелацију са академским успехом студената. Резултати су објављени у научном часопису *IEEE Access* 2024. године, са насловом *"Uncovering Determinants of Code Quality In Education Via Static Code Analysis*" [17]. Описана фаза и њено истраживање започето је 2021. године са прегледом литературе у овој области. На основу прегледа истраживања, идентификовани су фактори који представљају потенцијалне одреднице квалитета кода. Идентификовани фактори коришћени су у истраживању како би се утврдиле одреднице квалитета кода.

У последњој фази, истраживање је проширено на плагијаризам и аутоматско оцењивање студентских пројеката, укључујући развој нових модела за анализу и оцењивање, као и сагледавање како статичка анализа кôда може помоћи у овим процесима. Фаза такође обухвата спровођење упитника међу наставницима у области софтверског инжењерства за сакупљање података, који ће бити коришћени у истраживању. Као и и у претходној фази, и у овом делу истраживања најпре је спроведен преглед литературе у овој области, идентификовани су кључни чиниоци истраживања, затим је спроведен упитник над наставницима у области софтверског инжењерства који ће детаљније бити описан у дисертацији.

На слици 2 графички је представљен ток истраживања, са временским одредницама спровођења сваке фазе.



Слика 2 – Ток истраживања докторске дисертације

1.2 Очекивани резултати истраживања и научни допринос дисертације

Модел који је развијен и представљен у оквиру ове докторске дисертације интегрише статичку анализу кôда у процес оцењивања софтверских пројеката, где се поред процене различитих аспеката квалитета кôда, укључује и идентификација плагијаризма.

Научни доприноси укључују:

- Преглед могућности и потенцијала алата заснованих на статичкој анализи кôда: Прегледом постојеће литературе у области, упитником и спроведеним истраживањем над студентским пројектима, дисертација обухвата, користи и истиче све могућности и предности примене алата за статичку анализу кôда.
- Идентификацију и анализу кључних одредница квалитета кôда: Интеграцијом свих варијабли истакнутих у прегледу стања у области, дисертација обухвата широк спектар фактора који утичу на квалитет кôда, за разлику од претходних истраживања која свој фокус усмеравају на појединачне одреднице.
- Развој модела оцењивања софтверских пројеката: Модел обезбеђује објективност, тачност и академски интегритет у процесу оцењивања студентских пројеката, интегришући идентификоване одреднице квалитета кôда и алате за статичку анализу и детекцију плагијаризма.

Апликативни доприноси обухватају:

- Формулисање смерница и препорука за образовне институције: Дисертација пружа практична упутства о томе како ефикасно имплементирати алате за статичку анализу кôда и методе за детекцију плагијаризма у наставне активности, са циљем унапређења процеса оцењивања и подизања квалитета образовања.
- Опис корака за организацију и поставку софтверских пројеката: Пружене су смернице које омогућавају да се варијабле које су се истакле као одреднице квалитета кôда примене на начин који подстиче развој софтверских пројеката највишег квалитета.

Друштвени доприноси обухватају:

- Унапређење образовања у области софтверског инжењерства: Предложени модел и препоруке омогућавају студентима да стекну дубље разумевање принципа квалитетног кодирања, што доприноси њиховој професионалној спремности за рад у индустрији.
- Објективно оцењивање студентских пројеката: Применом статичке анализе кôда и детекције плагијаризма обезбеђује се праведно и доследно оцењивање, што подстиче студенте да улажу већи труд у израду својих пројеката.
- Олакшан процес оцењивања за наставно особље: Аутоматизација процеса оцењивања помоћу развијеног модела и алата смањује време и напор потребан за преглед великог броја студентских пројеката, што наставницима омогућава да се више фокусирају на едукативни аспект наставе.
- Промоција академског интегритета: Интеграцијом метода за детекцију плагијаризма у процес оцењивања, дисертација доприноси стварању академског окружења које подстиче оригиналност, поштење и професионалну етику код студената.
- Развој висококвалитетних софтверских решења: Применом идентификованих одредница квалитета кôда и побољшањем образовног процеса, студенти се охрабрују

да развијају софтверске пројекте који су технички исправни, одрживи и релевантни за потребе индустрије.

План даљег истраживања укључује:

- Развој алата који би подржао предложени модел оцењивања пројеката: Ово би омогућило аутоматизацију процеса оцењивања, повећало ефикасност и обезбедило конзистентност у примени модела.
- Укључивање већег броја образовних институција у истраживање: Циљ је додатно испитати одреднице квалитета кôда и проблем плагијаризма, што би допринело повећању валидности резултата.
- Поређење добијених резултата са реалним пројектима из индустрије: Ово би помогло у утврђивању релевантности предложеног модела у практичном контексту и омогућило прилагођавање модела потребама индустрије.
- Прилагођавање модела за друге типове софтверских пројеката: Ово би осигурало флексибилност и применљивост модела у различитим областима софтверског инжењерства и образовања.

Кроз ове научне и апликативне доприносе, дисертација значајно обогаћује област оцењивања софтверских пројеката. Комбинујући теоријска сазнања са практичним смерницама, пружа се основа за унапређење квалитета образовања, промовише академски интегритет и подстиче развој висококвалитетних софтверских решења.

1.3 Структура дисертације

Кроз реализацију фаза истраживања представљених у току истраживања докторске дисертације, обликована је структура дисертације сачињена од седам поглавља која су представљена у наставку.

Поглавље 1 – Уводна разматрања: У овом поглављу пружен је увид у основну мотивацију за истраживање, разматрајући значај, потребу и потенцијал примене алата за статичку анализу кода у академским (студентским) и другим типовима софтверских пројеката. Приказано је како примена алата за статичку анализу кôда може побољшати квалитет образовања и студентских пројеката у области софтверског инжењерства. У уводним разматрањима описан је и ток спровођења истраживања у склопу дисертације са нагласком на спроведене фазе и објављене публикације . На крају поглавља, представљени су очекивани научни, апликативни и друштвени доприноси дисертације и описана је структура дисертације.

Поглавље 2 – Теоријске основе: Описани су кључни теоријски концепти и методологије у вези са статичком анализом кôда, као и преглед примене алата и техника у индустрији и академским институцијама. Анализиран је претходни рад у овој области, са освртом на изазове и ограничења садашњих пракси. Разматран је и значај етике и интегритета у академском образовању, посебно у контексту плагијаризма и оцењивања студентских пројеката. На крају, дефинисана су истраживачка питања, хипотезе и општи циљ истраживања.

Поглавље 3 – Методолошки аспекти истраживања: У овом поглављу описане су истраживачке методе коришћене за прикупљање и анализу података, укључујући квантитативне и квалитативне технике. Представљени су инструменти за сакупљање података, као што су анкете и софтверски алати, као и методе за анализу података које обухватају статистичку обраду и интерпретацију резултата.

Поглавље 4 – Резултати истраживања: У овом поглављу представљена је детаљна анализа резултата истраживања, укључујући анализу добијених података из анкета и

софтверске анализе. Разматрани су различити аспекти утицаја статичке анализе кôда на квалитет студентских кôдова и пројеката, као и потенцијал за унапређење наставних метода и материјала.

Поглавље 5 – Дискусија резултата истраживања: Ово поглавље фокусирано је на тумачење резултата у контексту постављених истраживачких питања и теоријске основе. Оцењује се значај истраживања за академску заједницу и предлажу се промене у пракси које могу помоћи у побољшању квалитета кôда. Дискусија такође покрива ограничења истраживања, могућности примене предложених модела на различите типове софтверских пројеката и потенцијал за будућа истраживања у области.

Поглавље 6 – Закључна разматрања и правци даљег истраживања: Ово поглавље садржи главне закључке истраживања, утицај на теоријске и практичне аспекте у образовању и предлаже детаљне смернице за имплементацију статичке анализе кôда у образовне процесе. Такође, разматрају се препоруке за будућа истраживања и потенцијал примене предложеног модела.

Поглавље 7 – Литература: Садржи списак свих научних и стручних радова, књига, чланака и других ресурса који су коришћени или цитирани током израде дисертације. Овај списак обезбеђује основу за даље истраживање и потврду коришћених извора.

На крају докторске дисертације приложена је библиографија примарних студија, упитник, коришћени изворни кôдови и преглед опсервација.

2. Теоријске основе

У оквиру овог поглавља најпре су дефинисани основни концепти статичке анализе кôда, чије је познавање неопходно за разумевање методологије и резултата истраживања. Након тога, следи преглед стања у области, који приказује резултате систематских прегледа литературе спроведених ради утврђивања кључних фактора истраживања. У циљу утврђивања потреба за истраживањем, приказани су резултати претходних истраживања која укључују постојеће методе и технике примене алата за статичку анализу кôда. У склопу описа теоријских основа поред прегледа литературе, описано је и спроведено емпиријско истраживање које испитује примену статичке анализе кôда у индустрији. На крају су представљена два истраживачка модела са хипотезама, дефинисана на основу теоријских основа.

2.1 Основни концепти статичке анализе кôда

У овом потпоглављу је описан и истакнут значај квалитета кôда, основни концепти и дефиниција статичке анализе кôда. Поред тога, описане су и технике и методе спровођења статичке анализе, као и начин функционисања и могућност примене алата за статичку анализу кôда.

2.1.1. Дефиниција и значај квалитета кôда

Квалитет кôда представља основни показатељ исправности и ефикасности софтвера, фокусирајући се на то колико добро кôд задовољава предефинисане функционалне и непрограмске захтеве. Како би кôд био високог квалитета, мора бити читљив, одржив, перформантан, сигуран и преносив [18]. Читљивост подразумева лакоћу са којом други програмери могу да разумеју и интерпретирају кôд, док одрживост указује на могућност лаког ажурирања и модификације кôда без значајних препрека или ризика од увођења нових грешака [1].

Квалитет кôда је од критичне важности за успешност било ког софтверског пројекта. Висок квалитет кôда значајно смањује време потребно за одржавање и исправке, чиме се омогућава тимовима да се фокусирају на додавање нових функција, уместо на решавање проблема. Поред тога, добро написан кôд увећава тимску продуктивност, јер смањује потребу за дугим сесијама дебаговања и омогућава лакшу колаборацију међу програмерима. Такође, квалитет кôда директно утиче на укупне трошкове развоја софтвера, смањујући их кроз умањење потребе за честим изменама и усклађивањима [1], [5].

Неколико кључних фактора утиче на квалитет кôда. Пре свега, стил кодирања, који подразумева конзистентност у називима, употреба шаблона и примена добрих пракси које имају значајан утицај на читљивост и одрживост кôда [5]. Архитектура софтвера, укључујући начин на који су компоненте организоване и интегрисане, директно утиче на његову скалабилност и ефикасност [19]. Тестирање је још један виталан елемент који помаже у осигуравању да кôд функционише како је предвиђено, док су алати за анализу кôда незаменљиви у идентификацији потенцијалних проблема пре продукционисања. Управо ови алати омогућавају аутоматизовану проверу квалитета и усклађености са стандардима, што је од пресудног значаја у савременим развојним окружењима [20].

Оваквим уводом у квалитет кôда, постављени су темељи за детаљније истраживање специфичних аспеката и техника које се користе за његово осигурање, нарочито кроз статичку анализу кôда, о којој ће бити више речи у наставку.

2.1.2. Статичка анализа кôда

Статичка анализа кôда представља метод који омогућава анализу кôда без потребе за његовим извршавањем. Овај метод се користи за проналажење грешака, рањивости, и других потенцијалних проблема у кôду, само на основу његове структуре и садржаја [1]. За разлику од динамичке анализе, која укључује извршавање кôда у контролисаном окружењу, статичка анализа кôда користи различите технике за инспекцију кôда у циљу идентификације неправилности које могу указивати на потенцијалне грешке или безбедносне ризике. Динамичка анализа је веома важна област анализе кôда и део је многих истраживања [21], [22], [23], [24], [25]. Међутим, она није предмет овог истраживања и неће бити детаљније разматрана.

Главни циљеви статичке анализе укључују неколико кључних аспеката [5]:

1. Откривање грешака: Статичка анализа може ефикасно идентификовати синтаксне и семантичке грешке пре него што кôд уђе у фазу тестирања или продукционисања.
2. Побољшање читљивости: Кроз анализу и сугестије за измене, ова техника може унапредити структуру и стил кôда, чинећи га доступнијим и разумљивијим за остале развојне тимове.
3. Идентификација сигурносних рањивости: Статичка анализа је веома важна у процесима осигурања безбедности софтвера, јер може упозорити на потенцијалне угрожености пре него што постану проблем.
4. Оптимизација перформанси: Анализирањем кôда, могу се открити и исправити делови кôда који не делују ефикасно, што доводи до побољшања укупних перформанси софтвера.

Предности статичке анализе укључују њену способност да брзо и ефикасно идентификује грешке које би могле бити прескочене током мануелног прегледа кôда. Аутоматско проверавање стандарда кодирања је још једна значајна предност, јер омогућава одржавање конзистентност и квалитета кôда у пројектима на високом нивоу.

Са друге стране, статичка анализа носи са собом и нека ограничења, укључујући потенцијал за генерисање лажно позитивних резултата, који могу довести до непотребног прегледа неправилности које заправо не постоје [12]. Такође, статичка анализа кôда не може у потпуности заменити потребу за динамичком анализом, јер неке грешке и рањивости могу бити откривене само током стварног извршавања кôда.

На слици 3 је представљена класификација типова неправилности које статичка анализа може да идентификује, као и њихов однос према видљивости у кôду или у дизајну. Графички приказ илуструје разлику између генеричких неправилности, које алати лако препознају, и неправилности специфичних за контекст, које захтевају дубље разумевање доменске логике и принципа безбедности.

Слика 3 – Генеричке неправилности и неправилности специфичне за контекст [1]

Статичка анализа кôда представља незаменљив метод у софтверском инжењерству, који помаже у осигурању да кôд функционише како је предвиђено и да буде усклађен са најбољим праксама и стандардима индустрије.

2.1.3. Технике и методе статичке анализе кôда

Технике статичке анализе кôда развијене су као одговор на потребу за раном детекцијом грешака у процесу развоја софтвера, чиме се смањују трошкови и време потребно за исправке у каснијим фазама. Ове технике произилазе из области теорије формалних језика, где је анализа синтаксе и семантике програмских језика била основа за развој алата који могу аутоматски анализирати кôд [26]. Статичка анализа се развијала паралелно са напретком у компјутерској науци, усвајајући сложеније алгоритме и методе за обраду кôда који постају све обимнији и сложенији.

С временом, различите технике су се специјализовале за разне аспекте анализе кôда, доводећи до креирања детаљне категоризације метода које адресирају синтаксне, семантичке, и логичке аспекте програма. Ове методе су класификоване у неколико основних категорија које укључују анализу синтаксе, семантике, контролног тока, као и анализу тачности и потпуности [5]. Комбинацијом описаних техника, статичка анализа омогућава да се кôд прегледа и оптимизује на начин који би традиционалним путем био готово неизводљив због велике комплексности и обима модерних софтверских система.

Синтаксна анализа представља основну технику у статичкој анализи кôда, која проверава да ли кôд поштује дефинисана синтаксна правила програмског језика. Ова анализа је критична јер грешке у синтакси могу довести до грешака у компајлирању или непредвидивог понашања програма. Коришћењем ове анализе, развојни тимови могу брзо идентификовати и исправити основне грешке у кôду пре него што он уђе у даље фазе тестирања или продукционисања, што значајно штеди време и ресурсе.

Семантичка анализа се бави интерпретацијом значења кôда у контексту који дефинишу правила програмског језика. Ова анализа укључује проверу исправности типова података и обезбеђивање коректног дефинисања и коришћења свих променљивих и свих функција. Семантичка анализа такође може открити грешке у логици које могу изазвати непредвидиве или нежељене операције у програму, што помаже у повећању поузданости и сигурности софтвера [27].

Анализа контролног тока проверава логички ток извршења програма, утврђујући могуће начине којим може бити извршен кôд. Ово је посебно важно за идентификацију „мртвог кôда“ (кôд који се никада не извршава) и потенцијалних, бесконачних петљи. Такође, фокусира се и на праћење тока података кроз програм, осигуравајући да су све променљиве иницијализоване пре употребе и да не постоје непредвиђене или небезбедне измене података, које могу угрозити стабилност или безбедност програма [28].

Технике анализе тачности и потпуности укључују идентификацију логичких грешака и недоследности у кôду, које могу довести до непредвиђених резултата. Описане технике помажу развојним тимовима да верификују да ли програм ради како је предвиђено у свим могућим условима и да ли су сви могући случајеви употребе адекватно покривени. Овим се значајно повећава укупна поузданост и квалитет софтверског производа [29].

Свака од описаних техника статичке анализе кôда игра кључну улогу у развоју поузданих, безбедних и ефикасних софтверских апликација. Комбиновањем техника, тимови могу да осигурају функционалност свог кôда, као и његову оптимизацију за једноставно одржавање и скалабилност.

2.1.4. Алати за статичку анализу кôда

Статичка анализа кôда користи различите алате који су развијени како би аутоматизовали и систематизовали процес провере квалитета кôда. Алати су настали као директан резултат потребе за ефикаснијим и поузданијим методама верификације кôда у току брзог развоја софтверских технологија и повећања сложености апликација [1]. Алати за статичку анализу обично се интегришу у развојна окружења и процесе континуиране интеграције, омогућавајући развојним тимовима да непрекидно проверавају и унапређују квалитет свог кôда.

Међу најпознатијим алатима за статичку анализу кôда се налазе [12]:

1. *SonarQube*: Омогућава детаљну анализу кôда, откривање грешака, рањивости и *code smells*. Подржава већи број програмских језика [30].
2. *PMD*: Фокусиран на откривање потенцијалних грешака као што су неиспуњени дизајнерски обрасци, неефикасне методе, и сложен кôд [31].
3. *ESLint*: Специјализован за *JavaScript* програмски језик, овај алат помаже у идентификацији проблематичних образаца у кôду и подржава конфигурисање правила анализе [32], [33].
4. *FindBugs*: Фокусиран на *Java* програмски језик, овај алат користи статичку анализу за откривање грешака у бајт-кôду које могу указивати на уобичајене програмске грешке [34].

Ови алати биће детаљније описани у поглављу 3.1.5.

Алати за статичку анализу функционишу тако што парсирају изворни кôд и генеришу апстрактна синтаксна стабла која представљају структуру кôда. Затим примењују низ правила и образаца на ове структуре у потрази за познатим грешкама или проблемима. Анализа укључује проверавање стандарда кодирања, понашање података и логичке токове, а резултати се често приказују у виду извештаја који помажу софтверским инжењерима да увиде и исправе проблеме.

У пракси, алати за статичку анализу су интегрисани у развојне процесе тако да аутоматски проверавају кôд при свакој измени у репозиторијуму или током процеса продукционисања апликације. Ово омогућава тимовима да одржавају висок квалитет кôда, умањујући ризик од увођења грешака које могу ометати продукционисање или компромитовати сигурност софтвера. Кроз коришћење алата, развојни тимови могу значајно побољшати поузданост и безбедност својих софтверских производа, што у крајњој линији води ка стварању квалитетнијих и поузданијих апликација.

2.1.5. Стандард квалитета софтверског производа – *ISO/IEC* 25010:2023

ISO/IEC (*International Organization for Standardization/International Electrotechnical Commission*) 25010:2023 представља важан стандард у области системског и софтверског инжењеринга, који дефинише моделе квалитета система и софтвера. Овај стандард је део ширег скупа стандарда познатог као Системи и софтверско инжењерство – Захтеви и евалуација квалитета система и софтвера (енгл. *Systems and Software Quality Requirements and Evaluation – SQuaRE*). Његова примарна улога је да пружи оквир за процену квалитета софтверских производа и система, чиме се омогућава објективно мерење и процена квалитета кроз различите аспекте и карактеристике [35].

ISO/IEC 25010:2023 дефинише два главна модела квалитета:

1. модел квалитета производа и
2. модел квалитета употребе.

Оба модела нуде детаљан поглед на различите карактеристике које је потребно размотрити приликом евалуације квалитета софтвера.

Модел квалитета производа обухвата карактеристике које описују унутрашњи и спољашњи квалитет софтверског производа. Унутрашњи квалитет се односи на структуру и сам кôд производа, док спољашњи квалитет мери начин на који софтверски производ функционише у стварним условима. Карактеристике овог модела укључују [35]:

- Функционалну погодност: Способност софтвера да задовољи функционалне захтеве.
- Перформансе ефикасности: Односи се на време одзива система и употребу ресурса.
- Компатибилност: Способност софтвера да функционише у различитим окружењима или са другим системима.
- Употребљивост: Лакоћа коришћења система од стране корисника.
- Поузданост: Степен до ког систем може да обавља своје функције под датим условима.
- Сигурност: Способност система да заштити податке и одржи поверљивост.
- Одрживост: Могућност лаког прилагођавања и одржавања система.
- Преносивост: Способност софтвера да се лако пребацује у различита окружења.

Модел квалитета употребе се фокусира на начин на који корисници перципирају квалитет употребе софтвера у специфичном контексту. Он мери у којем обиму софтвер испуњава корисничке потребе и омогућава постизање задатих циљева. Кључне карактеристике укључују [35]:

- Ефективност: Способност корисника да постигну своје циљеве коришћењем система.
- Ефикасност: Ниво продуктивности који се постиже коришћењем система.
- Задовољство корисника: Мера задовољства корисника функционалношћу и једноставношћу коришћења система.
- Слобода од ризика: Степен у којем систем омогућава безбедну и поуздану употребу.
- Контекст покривености: Способност софтвера да функционише у различитим корисничким сценаријима.
- Примена стандарда у евалуацији софтвера.

Примена овог стандарда омогућава систематизован приступ процени квалитета софтвера. Компаније и организације које развијају софтвер могу користити овај оквир како би осигурале да њихови производи задовољавају захтеве корисника и да су усклађени са индустријским стандардима. У пракси, *ISO/IEC* 25010:2023 помаже у дефинисању квалитета у раним фазама развоја софтвера, а такође пружа основе за независну валидацију и верификацију квалитета производа.

ISO/IEC 25010:2023 представља значајан допринос развоју квалитетног софтвера, омогућавајући јасне критеријуме за процену и побољшање како техничких, тако и корисничких аспеката система.

2.2 Класификација софтверских пројеката

Софтверски пројекти представљају комплексне активности које укључују планирање, развој, тестирање и одржавање софтверских система. Разумевање различитих типова софтверских пројеката је од суштинског значаја за примену одговарајућих метода и алата за њихово успешно извођење и оцењивање квалитета [36], [37]. У овој секцији представљена је класификација софтверских пројеката која обухвата четири основне категорије типова софтверских пројеката [36], [37]:

- академски (студентски) софтверски пројекти,
- комерцијални (индустријски) софтверски пројекти,
- отворени (енгл. *open source*) софтверски пројекти и
- државни (јавни) софтверски пројекти.

Академски софтверски пројекти су пројекти које развијају студенти или истраживачи у оквиру образовних институција. Наведених пројекти служе као практична примена теоријског знања и омогућавају студентима да стекну искуство у софтверском инжењерству [37]. Карактеристике академских пројеката укључују ограничен обим, временска ограничења услед академског календара и фокус на процес учења више него на сам финални производ.

Комерцијални софтверски пројекти су пројекти које развијају компаније за тржиште или за интерну употребу. Они су усмерени на задовољавање потреба клијената, профитабилност и одрживост [36]. Пројекти се одликују високим стандардима квалитета, комплексношћу, великим тимовима и потребом за поштовањем пословних и правних регулатива.

Отворени софтверски пројекти су пројекти чији је изворни кôд доступан јавности под одређеним лиценцама. Развијају их заједнице програмера из целог света, често на волонтерској основи [37]. Карактеришу их транспарентност, колаборативни развој и могућност прилагођавања софтвера специфичним потребама корисника.

Државни или јавни софтверски пројекти су пројекти које развијају или наручују државне институције за потребе јавних служби или интерну употребу. Пројекти су усмерени на поузданост, безбедност и доступност, и често имају дугорочан утицај на друштво [36]. Они захтевају поштовање строгих стандарда и регулатива.

Иако сваки од наведених типова софтверских пројеката има своје специфичности и изазове, ово истраживање фокусирано је на академске (студентске) софтверске пројекте.

Студентски пројекти представљају погодну основу за развој и тестирање модела оцењивања заснованог на статичкој анализи кôда, јер омогућавају систематско прикупљање и анализу података у контролисаном окружењу. Иако ће модел бити развијен и примењен у контексту академских пројеката, у раду ће бити описана и могућност адаптације на друге типове софтверских пројеката. Ово отвара простор за будућа истраживања и потенцијално ширење примене модела у комерцијалним и другим контекстима.

2.3 Преглед стања у области

Преглед стања у области приказује резултате систематских прегледа литературе спроведених ради утврђивања фактора истраживања. Они се тичу употребе алата за статичку анализу кôда, примене таквих алата у индустрији и образовању, фактора који утичу на квалитет кôда, примене статичке анализе кôда у детекцији плагијаризма и за потребе аутоматског оцењивања софтверских пројеката.

2.3.1. Преглед употребе алата за статичку анализу кôда

Први преглед стања у области који је спроведен за потребе овог истраживања имао је за циљ да прикупи информације о употреби алата за статичку анализу кôда, издвоји алате који се најчешће користе у истраживањима и класификује типове неправилности које су алати у стању да детектују. Поред наведеног, овим прегледом обухваћено је и испитивање могућности примене алата за статичку анализу кôда над језицима наменским за домен. Такође, издвајане су и информације за које све програмске језике опште намене (енгл. *general-purpose languages*) алати обезбеђују подршку. Детаљна процедура и резултати анализе су објављени на међународној *DAAAM* конференцији [12], а у даљем тексту биће представљен сажетак.

У складу са процедуром систематског прегледа литературе, који предлаже Барбара Киченхам [38], овај преглед је подељен у три основне фазе: планирање прегледа, спровођење прегледа и извештавање о прегледу.

Протокол наведеног систематског прегледа литературе подразумева најпре дефинисање истраживачких питања, термина претраге и критеријума укључивања и искључивања. Формулисана су следећа истраживачка питања:

ИП1: Да ли постоје алати за статичку анализу кôда применљиви на било који програмски језик опште намене?

ИП2: Да ли постоји алат за статичку анализу кôда који обезбеђује подршку за језике наменске за домен?

ИП3: Који су алати најчешће помињани у истраживачким студијама?

ИП4: Да ли постоји алат који открива све типове неправилности?

За потребе овог прегледа литературе, претрага је обављена у базама података *SCOPUS* и *Google Scholar*, користећи термине претраге: "*Static code analysis*" *AND* "*tools*" *AND* "*detection*" и година објављивања >= 2010.

Претрага је резултирала са 346 радова.

Критеријуми укључивања су:

- **КУ1:** Рад мора бити написани на енглеском језику.
- **КУ2:** Рад мора представљати алате засноване на статичкој анализи кôда.
- **КУ3:** Рад мора представити подржане програмске језике од стране алата за статичку анализу кôда.

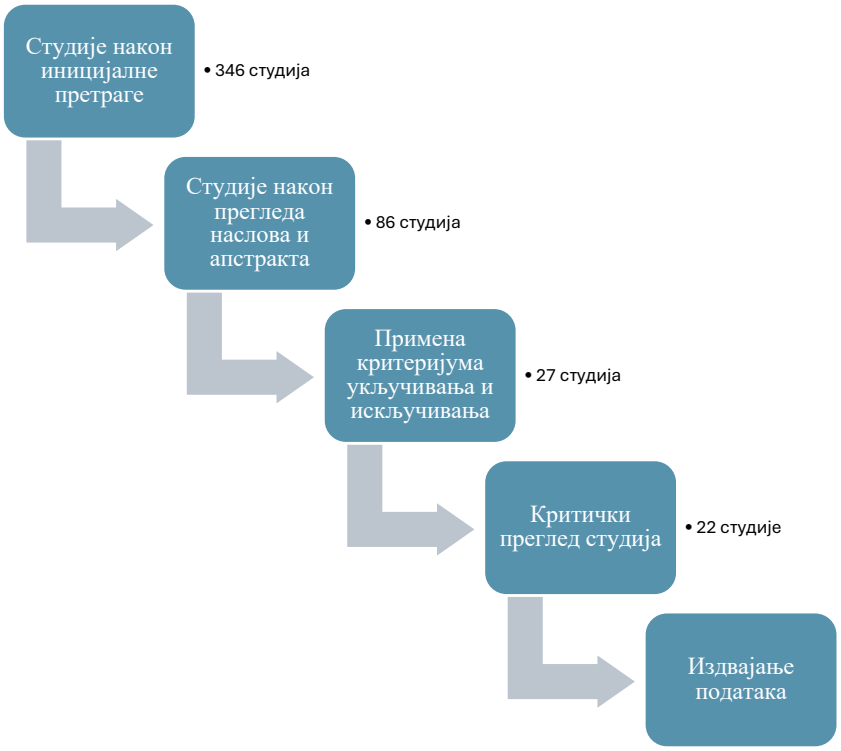
Критеријуми искључивања су:

- **КИ1:** Све дупликације радова.
- **КИ2:** Уколико аутор има више од једног рада о истом алату, само један рад би требало да буде укључен.

Приликом прегледа студија издвајане су и бележене следеће информације:

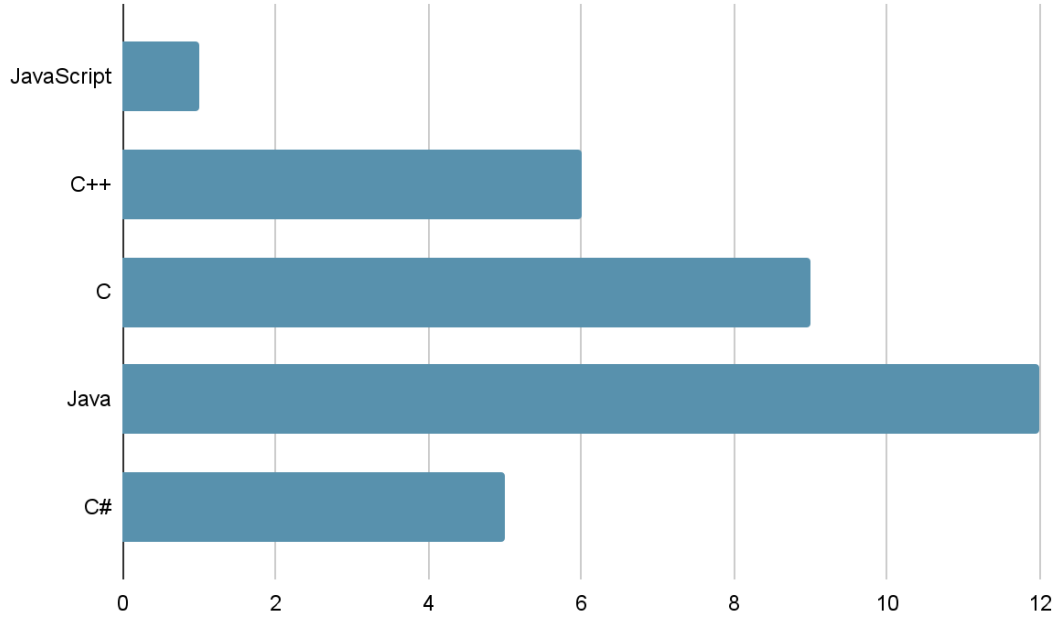
1. година објављивања и тип публикације,
2. предложен алат,
3. типови неправилности које детектују и
4. подржани програмски језици.

Ток спровођења систематског прегледа литературе са бројем радова након сваке фазе приказан је на слици 4.



Слика 4 – Ток спровођења прегледа стања у области

У протоколу прегледа формулисана су три истраживачка питања. Слика 5 приказује издвојене податке који одговарају на прво истраживачко питање — **ИП1: Да ли постоје алати за статичку анализу кôда применљиви на било који језик опште намене?**



Слика 5 – Расподела примарних студија које описују алате са подршком за језике опште намене

Неки од представљених алата за статичку анализу кôда имају подршку за неколико језика опште намене, док други алати имају подршку само за један програмски језик. Већина алата за статичку анализу кôда подржава оне програмске језике који се најчешће користе у индустрији. Разлог за ово је то што на овај начин креатори алата за статичку анализу кôда обезбеђују већи број корисника свог алата.

Табела 1 представља издвојене податке који одговарају на друго истраживачко питање **— ИП2: Да ли постоји алат за статичку анализу кôда који се примењује на језике наменске за домен?**

У оквиру одабраних студија у прегледу литературе, само три студије (14%) представљају алате и подршку за језике наменске за домен. Насупрот томе, остали алати (86%) у студијама анализирају тачност и ефикасност алата за статичку анализу кôда који имају подршку за један или више језика опште намене.

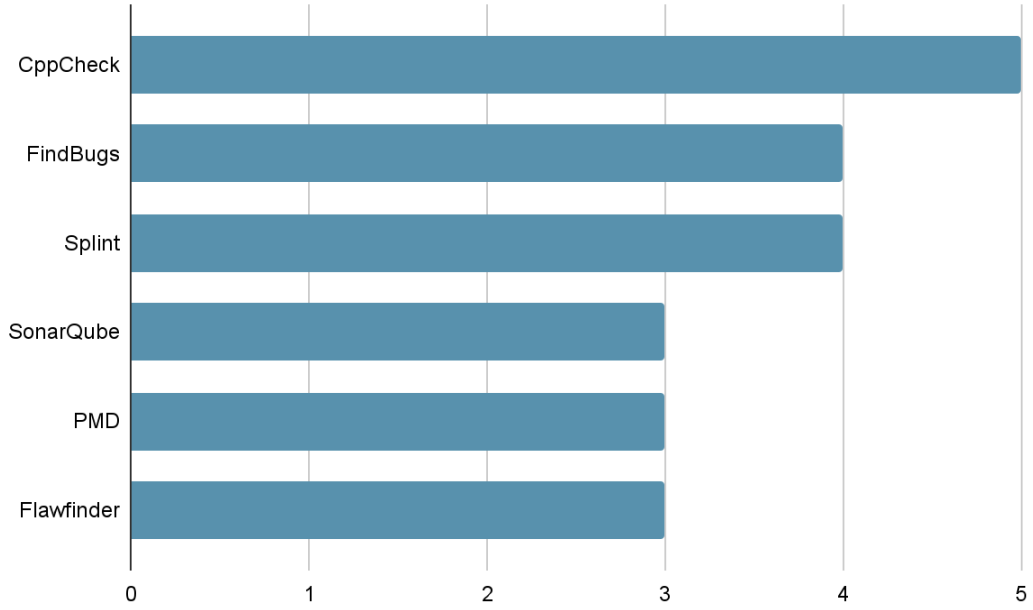
Табела 1 – Расподела примарних студија према типу језика за које алати обезбеђују подршку

Тип програмског језика	Примарне студије	%
Програмски језик наменски за домен	[39], [40], [41]	14%
Програмски језик опште намене	[39], [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [57], [58], [59], [60]	86%

Слика 6 представља издвојене податке који одговарају на треће истраживачко питање **— ИП3: Који алати се најчешће помињу у примарним студијама?**

Представљено је шест алата који су најчешће анализирани у примарним студијама. Алати су приказани три или више пута у различитим истраживањима. За разлику од тога,

остали алати који су представљени у студијама укљученим у овај преглед стања у области анализирани су у две или једној студији.



Слика 6 – Алати који су највише заступљени у примарним студијама

У табели 2 приказани су издвојени подаци који одговарају на истраживачко питање **— ИП4: Да ли постоји алат који открива све типове неправилности?**

Проблем који је примећен при издвајању података у вези са типовима неправилности које би требало открити је да студије не истичу типове неправилности које предложени алати откривају јасно и прецизно. Неке студије не истичу типове неправилности које предложени алати откривају на разумљив начин. Такве студије [40], [41], [52], [55], [58] су искључене из ове класификације по типу неправилности.

Други примећени проблем је да студије користе различите таксономије за класификацију неправилности. У овом истраживању успостављена је јединствена таксономија, која се назива „Седам погубних краљевстава” (енгл. *Seven Pernicious Kingdoms*) [61], и сви типови неправилности које различите студије истичу класификовани су у складу са њом.

Табела 2 – Примарне студије класификоване према типу неправилности које представљени алати детектују

Тип неправилности	Примарне студије
Валидација и представљање уноса	[39], [42], [43], [44], [45], [46], [47], [48], [49], [54], [57], [59], [60]
API abuse	[46], [54], [56], [60]
Безбедност	[42], [46], [47], [51], [53], [54], [60]
Време и стање	[46], [47], [55], [57]
Грешке	[42], [45], [46], [47], [49], [54], [57]
Квалитет кôда	[39], [42], [46], [47], [49], [54], [56], [57]
Encapsulation	[46]
Окружење	[54]

Резултати овог систематског прегледа литературе показују да алати за статичку анализу кôда играју значајну улогу у савременом развоју софтвера, посебно у побољшању квалитета изворног кôда. Највећи број радова објављен је 2012. и 2016. године, што указује на периоде повећаног интересовања за ову тему.

Истраживање са Универзитета Алто истиче се као посебно значајан допринос овој области, јер детаљно описује експерименте спроведене на представљеним алатима. Овај рад нуди увид у практичну примену алата и њихов утицај на квалитет софтверских решења.

Одговарајући на прво истраживачко питање (ИП1), утврђено је да сваки од разматраних алата има подршку за бар један језик опште намене, док неки алати, попут оних описаних у студији [58], подржавају више језика опште намене. Ово потврђује универзалност и флексибилност алата за статичку анализу кôда у различитим развојним окружењима.

У вези са другим истраживачким питањем (ИП2), анализом је потврђено да постоји више алата који пружају подршку за језике наменске за домен, чиме се наглашава важност развоја специјализованих алата који могу адресирати посебне потребе у различитим областима софтверског инжењерства. Ово указује на потенцијал за ширење примене статичке анализе кôда у специфичним технолошким и индустријским секторима.

На крају, када је реч о трећем истраживачком питању (ИП3), видљиво је да су неки алати, као што су *SonarQube*, *PMD* и *Checkstyle*, чешће представљани у студијама због своје ефикасности и широке примене. Овај налаз истиче значај избора адекватних алата за статичку анализу кôда који могу знатно утицати на смањење неправилности и побољшање сигурности софтверских производа.

Дискусија резултата доприноси дубљем разумевању тренутних трендова и изазова у области статичке анализе кôда, као и потреби за наставком истраживања у циљу развоја нових алата и техника које ће омогућити ефикасније управљање квалитетом кôда у будућности.

2.3.2. Преглед примене алата за статичку анализу кôда и одредница квалитета кôда

Други преглед стања у области који је спроведен за потребе ове дисертације имао је за циљ да прикупи информације о факторима који могу да имају утицај на квалитет кôда у софтверским пројектима, заступљености статичке анализе кôда у индустрији, али и њеној примени у образовању будућих софтвер инжењера. Преглед стања у области описан је у склопу секције ”Стање у области” у истраживачком раду објављеном у часопису *IEEE Access* [17], а у даљем тексту биће представљен сажетак.

Прва активност у фази планирања била је анализа потребе за систематским прегледом литературе и провером постојећих радова на ову тему. Формулисана су следећа истраживачка питања:

ИП1: У којој мери се алати за статичку анализу кôда користе у пројектима у индустрији?

ИП2: Да ли наставници користе алате за статичку анализу кôда у процесу едукације будућих софтвер инжењера?

ИП3: Који фактори имају утицај на квалитет кôда?

За потребе овог прегледа литературе, претрага је обављена у базама података *SCOPUS* и *Google Scholar*, користећи термине претраге: *"Static code analysis" AND "Determinants" AND "Code quality" AND “Metrics” AND “Education” OR “Industry”*.

Претрага је резултирала са 433 рада.

Критеријуми укључивања су:

- **КУ1:** Рад мора бити објављен у претходних 10 година.
- **КУ2:** Рад мора представљати употребу неког алата у индустрији или едукацији.
- **КУ3:** Рад мора представљати одреднице квалитета кôда.

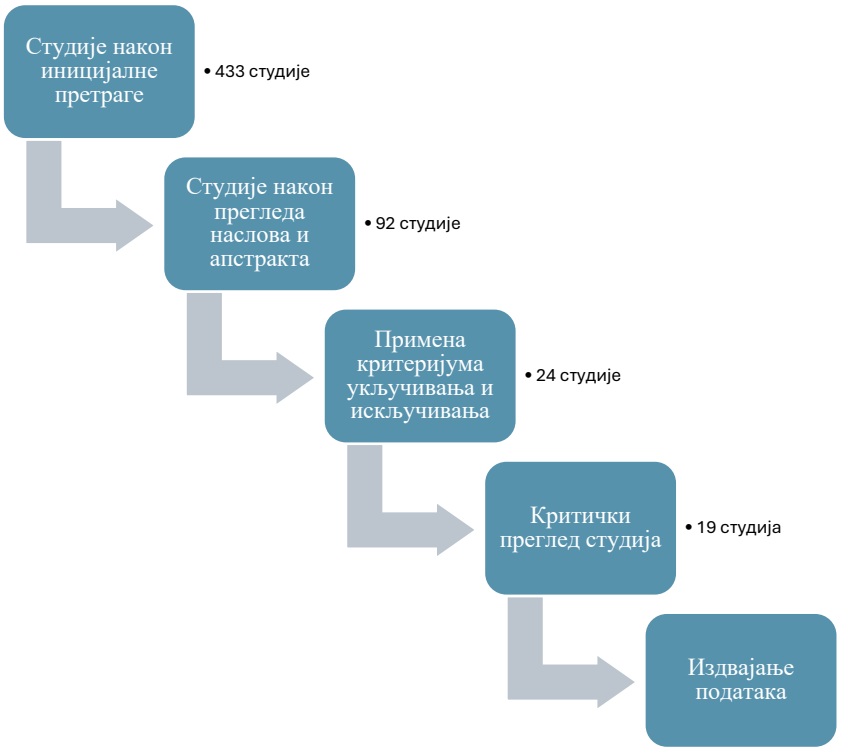
Критеријуми искључивања су:

- **КИ1:** Све дупликације радова.
- **КИ2:** Рад не описује употребу алата за статичку анализу кôда.

Приликом прегледа студија издвајане су и бележене следеће информације:

1. тип употребе,
2. фактори који утичу на квалитет кôда и
3. коришћени алати.

Ток спровођења систематског прегледа литературе са бројем радова након сваке фазе приказан је на слици 7.



Слика 7 – Ток спровођења другог прегледа стања у области

Нека истраживања обухваћена овим прегледом фокусирају се на новоразвијене алате за статичку анализу кôда. Студија [62] представља алат за статичку анализу програмског кôда програмабилних логичких контролера (енгл. *programmable logic controller – PLC*), детаљно описујући његову примену и резултате које постиже у индустријским пројектима. Емпиријска студија која илуструје коришћење статичке анализе кôда у индустрији и спремност програмера различитих нивоа искуства да користе ове алате описана је у [16]. Исти рад такође демонстрира све учесталије коришћење алата као обавезног процеса развоја у индустрији. Студија [20] наглашава потребу за дубљим разумевањем и мерењем квалитета софтвера изван традиционалних метрика. Поред тога, ова студија пореди програмерске навике између професионалаца у индустрији и студената, истичући области у којима студенти чешће праве грешке које би могле угрозити квалитет кôда.

Даље, алати за статичку анализу кôда све више налазе своју примену у образовању. Истраживање [63] представља алат *CodeMaster* базиран на статичкој анализи кôда, који аутоматски оцењује студентске пројекте, наглашавајући значај таквих алата у подизању квалитета образовања будућих софтвер инжењера. Слична претходној, студија [64] расправља о коришћењу алата за аутоматску верификацију програмских задатака, истичући њихов значај у образовним апликацијама.

У [65], приказани су резултати експеримента у којем студенти самостално користе *PMD*, алат за статичку анализу кôда, у циљу исправке неправилности у својим пројектима. Резултати ове студије такође откривају да студенти у великој мери разумеју извештаје које генерише овај алат и имају способност да исправе те неправилности.

Студије [66], [67] представљају укључивање алата за статичку анализу кôда у образовни процес. Експеримент се састојао од две групе студената. Прва група је користила алат за статичку анализу кôда током рада на пројекту, док контролна група није користила алат за своје пројекте. Резултати су показали да су студенти који су користили алат за статичку анализу кôда постигли боље резултате и развијали софтвер вишег квалитета у поређењу са контролном групом.

У студији [10] описан је развој и имплементација система *Edgar* за аутоматску евалуацију софтверских пројеката. Овај систем омогућава објективну и ефикасну оцену великог броја студентских пројеката, као и благовремену и корисну повратну информацију студентима. У оквиру њихове студије, наглашава се значај објективности овог система у процесу оцењивања студентских пројеката у поређењу са традиционалним методама оцењивања, приказујући пример и резултате коришћења алата *Edgar* у образовном систему.

Аутори [68] истичу значај укључивања алата за статичку анализу кôда у образовни процес. Ово истраживање такође говори о потреби за увођењем нових метрика које би се користиле у анализи студентских пројеката у поређењу са индустријским пројектима. Осим тога, рад адресира различите утицаје на квалитет кôда међу студентима, при чему истраживање [69] истиче значај и утицај тимског рада на квалитет студентских пројеката. Ово истраживање је пример ефикасне примене алата у образовном процесу, посебно истичући могућност идентификације студената који мање доприносе током тимског рада и како се то одражава на квалитет кôда. Друга студија [44], која расправља о утицају тимског рада на резултате пројеката, такође говори о изазовима стварања групних пројеката, утицају величине пројекта на резултате и методама и резултатима оцењивања таквих пројеката.

Истраживање [70], који истражује метрику софтверског замагљивања кроз систематски преглед литературе, истиче значај ове метрике за квалитет кôда, као и потребу за стандардизацијом ове метрике. Таква стандардизација би олакшала њено укључивање у алате за статичку анализу кôда и омогућила шире студије о индустријским пројектима.

Укратко, значај коришћења алата за статичку анализу кôда огледа се у све већој потреби за побољшањем квалитета кôда у индустријским пројектима [16]. Као и у индустрији, значај побољшања квалитета кôда почиње да се примењује и у образовању будућих софтвер инжењера на различите начине. Нека истраживања [68] показују примере где студенти самостално примењују ове алате за исправку неправилности које открију на својим пројектима, док са друге стране, одређена истраживања [10], [63] предлажу системе и алате који омогућавају наставницима да ефикасније и објективније оцењују студентске пројекте. Истраживање [69] расправља о фактору тимских и индивидуалних пројеката, затим истраживање [71] истиче фактор избора технологије, док истраживање [66] описује коришћење алата за статичку анализу кôда током рада на пројекту, као и фактор који описује начин одржавања наставе [72]. Осим утицаја описаних фактора и других одредница на квалитет кôда, у склопу овог прегледа укључене су и студије које говоре о академској успешности студената, који се огледа у оцени на предмету и брзини полагања пројекта [73], [74].

Синтеза резултата добијених из анализе студија обухваћених у прегледу литературе приказана је у табели 3.

Табела 3 – Синтеза резултата другог прегледа стања у области

Област		**Примарне студије**
Статичка анализа кôда у индустрији		[16], [19], [62]
Статичка анализа кôда у образовању	Коришћена од стране професора	[10], [20], [63]
	Коришћена од стране студената	[64], [65], [66]
Одреднице квалитета кôда	Начин сарадње на пројекту	[44], [69]
	Избор технологије	[71]
	Величина пројекта	[75]
	Искуство	[76], [77]
	Употреба алата за верзионисање	[78], [79]
	Употреба алата за статичку анализу кôда	[66]
	Начин одржавања наставе	[72]

Поред утицаја тимског рада на резултате студентских пројеката, други испитиван фактор је избор технологије пројекта. Истраживање [71] показује да избор софтверских технологија може значајно утицати на квалитет изворног кôда.

Даље, студија [80] истражује утицај функционалних и објектно-оријентисаних парадигми на квалитет кôда, приказујући значајне разлике у погледу специфичних метрика квалитета кôда у својим налазима. Слично томе, истраживање [81] доноси резултате који воде до закључка да коришћење програмског језика *C++* уместо *C* може довести до побољшања у квалитету софтвера, смањењу комплексности, смањењу склоности грешкама и мањем напору одржавања.

Иако истраживање [72] разматра различите резултате које студенти постижу у окружењу онлајн образовања у поређењу са традиционалном учењем (уживо, у просторијама факултета), овај преглед литературе није идентификовао никакве студије које дискутују директан утицај различитих метода наставе на квалитет кôда у студентским пројектима.

Као одредница квалитета кôда, такође ће се разматрати коришћење алата за верзионисање. У том контексту, студија [78] демонстрира како коришћење алата за верзионисање користећи *Bitbucket* [82] доприноси разумевању и примени стандарда кодирања. Насупрот томе, истраживање [79] истиче да коришћење алата за верзионисање користећи *Github* [83] олакшава имплементацију статичке анализе кôда, чиме се осигурава виши квалитет изворног кôда пројекта.

Метрика искуства, односно ниво вештина софтвер инжењера истиче се као валидан фактор утицаја на квалитет кôда. Студије [76], [77] показују како искуство софтвер инжењера утиче на квалитет кôда који производе током развоја.

Истраживање [75] анализира како број линија кôда утиче на комплексност софтвера и закључује да повећање броја линија кôда у софтверском пројекту доводи до повећања комплексности. Са друге стране, то директно утиче на повећање броја грешака, као и на тешкоће одржавања, поузданост и перформансе софтвера.

Посебна категорија литературе укључена у овај преглед фокусира се на методе мерења студентских перформанси у различитим областима. Тако, студије [72], [73], [84] разматрају оцену коју студент постиже на курсу као индикатор академске успешности по завршетку тог курса. С друге стране, друга варијабла која указује на студентску успешност је да ли студент испуњава своје обавезе на време. Студије [72], [74] такође узимају метрику времена или придржавања рокова у погледу испуњавања обавеза као варијаблу која указује на перформансе студента.

Овај преглед стања у области истиче разноврсност алата и приступа у статичкој анализи кôда, чинећи очигледним да је фокус већине истраживања у овој области усмерен ка два главна аспекта: побољшању техничког квалитета софтверских пројеката и мерењу утицаја примене алата у образовне сврхе. Међутим, мало студија директно истражује утицај различитих аспеката на квалитет кôда и како, ако уопште, генерисани квалитет кôда утиче на академску успешност будућих софтверских инжењера, остављајући простор за даља истраживања у овој области.

2.3.3. Преглед примене статичке анализе кôда у детекцији плагијаризма и оцењивању

Трећи преглед стања у области има за циљ да пружи свеобухватан увид у тренутна истраживања у области детекције плагијаризма у изворном кôду и аутоматизованог оцењивања у образовним окружењима.

У наставку текста се налазе истраживачка питања на којима је базиран овај преглед стања у области.

ИП1: Које методе се најчешће користе за детекцију плагијаризма у кôду у образовним окружењима и које су њихове предности и мане?

ИП2: Како се алати за аутоматизовано оцењивање интегришу у образовни процес и који су њихови ефекти на учење и наставу?

Претраживање литературе је извршено у базама података *SCOPUS* и *Google Scholar*, коришћењем следећих кључних речи: *"Source code plagiarism detection" OR "Automated programming assessment" AND "Education" AND "Static code analisys" AND "Code quality".* Претрагом је идентификовано укупно 384 рада који су анализирани и класификовани према њиховом фокусу и доприносу теми.

Критеријуми укључивања:

- **КУ1:** Рад мора бити објављен у последњих 10 година.
- **КУ2:** Рад мора да се бави специфичним методама за детекцију плагијата или аутоматизованим оцењивање у образовању у области софтверског инжењерства.
- **КУ3:** Рад мора бити емпиријска студија која пружа квантитативне или квалитативне резултате.

Критеријуми искључивања:

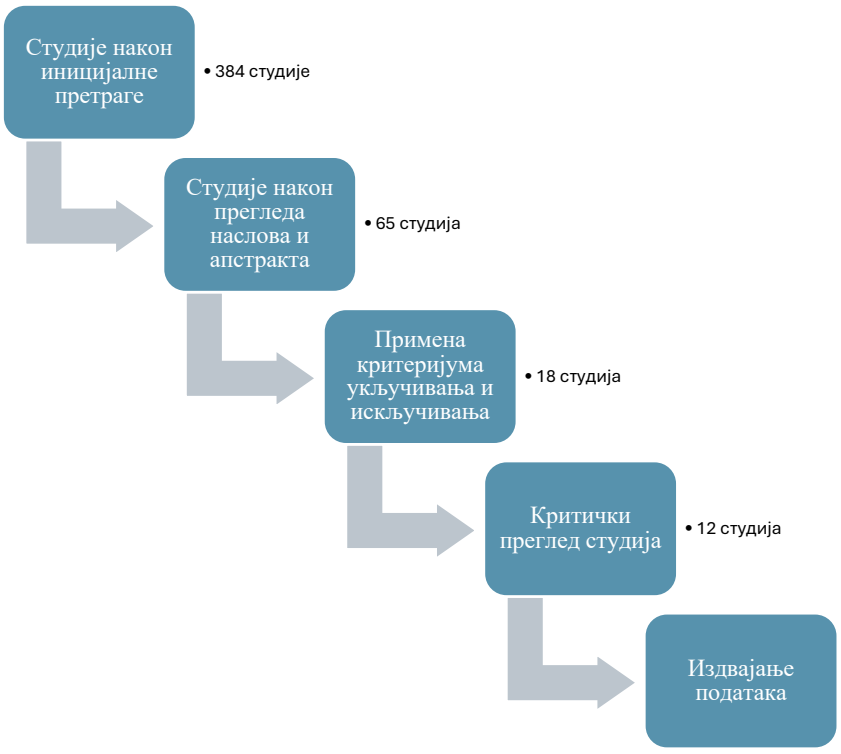
- **КИ1:** Сваки рад који не описује конкретне методе или алате за детекцију плагијата или аутоматизовано оцењивање.

- **КИ2:** Дупликати и радови који се фокусирају на опште теме без јасне примене у образовању или који не доприносе значајно унапређењу праксе или теорије у овој области.

Приликом прегледа студија издвајане су и бележене следеће информације:

1. тип употребе,
2. предности и недостаци употребе алата и
3. коришћени алати.

Ток спровођења прегледа стања у области са бројем радова након сваке фазе приказан је на слици 8.



Слика 8 – Ток спровођења трећег прегледа стања у области

Истраживање [9] испитује нови приступ који користи визуализацију мрежа сарадње на основу сличности између студентских решења програмског кôда. Овај приступ омогућава боље уочавање група фајлова који подлежу плагијаризму, а могли би бити занемарени у традиционалним табеларним приказима. Коришћењем алгоритама кластеровања, студија показује да је могуће идентификовати нове случајеве плагијата, што побољшава тачност и ефикасност у великом обиму података.

Студија [85] пружа детаљан преглед тренутног стања у области детекције плагијата програмског кôда на курсевима програмирања. Аутори су анализирали различите алате и технике који су развијени последњих година, укључујући анализу кôда, праћење историје предаја, *IP (*енгл. *internet protocol*) адресе и понашање студената, како би предложили робусније приступе за откривање плагијата.

Истраживање [86] се фокусира на употребу аутоматизованих техника за кластеровање кôда ради откривања плагијата у студентским задацима написаним у *Java* програмском језику. Студија користи алгоритам за израчунавање растојања уређивања стабла (енгл. *tree edit distance*) и технику *n*-грама за детекцију сличности између различитих решења, чиме се постиже висока прецизност у откривању плагијата.

Студија [87] истражује разумевање концепта плагијата међу студентима у области софтверског инжењерства у 18 институција у Великој Британији. Подаци сугеришу да постоје значајне разлике у разумевању између различитих демографских група, као и да постоје специфичне области у којима студенти нису сигурни шта се сматра прихватљивим или неприхватљивим понашањем, што има импликације за програме едукације о плагијату.

Публикација [88] анализира понашање студената у окружењу аутоматизованог оцењивања програмирања и утицај на квалитет кôда. Студија наглашава значај аутоматизованих алата за унапређење процеса учења и показује да овакав систем може побољшати разумевање програмских концепата кроз повратне информације које студенти добијају током решавања задатака.

Истраживање у склопу студије [10] разматра развој свеобухватног система за аутоматизовано оцењивање који подржава више програмских језика и типова питања. Овај систем омогућава аутоматско оцењивање студентских кôдова и анализу различитих аспеката квалитета кôда, укључујући стил, сложеност и перформансе, чиме се унапређује ефикасност образовног процеса.

Прегледом је обухваћен и систем *Edgar*, који интегрише интерактивне туторијале са аутоматизованим системима за оцењивање програма. Овај систем омогућава наставницима да креирају и дистрибуирају интерактивне туторијале који комбинују теоријско знање са практичним вежбама, пружајући повратне информације о грешкама у кôду у реалном времену.

У табели 4 приказан је преглед примарних студија према фокусу истраживања.

Табела 4– Синтеза примарних студија трећег прегледа стања у области према фокусу истраживања

Фокус истраживања	**Примарне студије**
Аутоматизовано оцењивање	[10], [11], [88], [89], [90], [91]
Плагијаризам	[8], [9], [85], [85], [86], [87]

Студија [11] представља ревизију система *Edgar*, која укључује нове модуле за интерактивно учење и оцењивање. Модули омогућавају аутоматско евалуирање различитих програмских језика и подршку за више типова питања, укључујући *multiple-choice* питања и кодирање базирано на игри, што побољшава процес учења и омогућава наставницима да ефикасније управљају наставним садржајем.

Истраживање [89] разматра развој и имплементацију туторијала базираних на облаку (енгл. *cloud-based*) који комбинују софтвер за биоинформатику, интерактивно кодирање и визуализацију. Овај приступ омогућава студентима да уче на даљину кроз интерактивне вежбе, чиме се побољшавају њихове практичне вештине и разумевање сложених концепата у биоинформатици.

Истраживање у склопу [90] истражује примену *peer assessment* метода у отвореним задацима у оквиру универзитетског курса. Аутори анализирају податке прикупљене током двогодишње студије, указујући на предности и изазове примене ове методологије, укључујући побољшање ангажованости студената и квалитета повратних информација.

Рад [8] анализира употребу стилских метрика за детекцију плагијата и процену ауторства у студентским задацима. Истраживање показује да анализа стилова кодирања може бити ефективан метод за диференцијацију између аутора и откривање сличности које традиционални алати можда не би идентификовали.

Рад [91] истражује примену аутоматизованих система оцењивања у *MOOC* (*Massive Open Online Courses*). Аутори анализирају различите сценарије за пружање практичних

програмских задатака уз аутоматизовано оцењивање, наглашавајући изазове и предности оваквог приступа у великом окружењу електронског учења. Студија такође разматра могућност примене различитих алата за аутоматизовано оцењивање у контексту *MOOC* платформи, истичући потребу за скалабилним решењима која омогућавају лакше и ефикасније учење.

У табели 5 приказана је синтеза примарних студија према циљној групи на коју се односи истраживање.

Табела 5 – Категоризација примарних студија трећег прегледа стања у области према циљној групи на коју се односи

Циљна група	Примарне студије
Студенти	[8], [9], [11], [86], [87], [88], [89], [90]
Наставници	[10], [11], [85], [91]

Преглед стања у области показује значајан напредак у области детекције плагијата програмског кôда и аутоматизованог оцењивања у образовним окружењима. Истраживања су усмерена ка побољшању ефикасности и објективности процеса оцењивања, као и ка развоју нових алата и техника који могу да подрже различите методе учења и наставе. Упркос постојећим унапређењима, и даље постоји простор за даља истраживања која би испитивала директан утицај алата на академску успешност и како они могу бити интегрисани у различита образовна окружења.

Будућа истраживања би требало да се фокусирају на дугорочне ефекте коришћења аутоматизованих алата, као и на развој нових метрика за процену квалитета кôда и учинка студената.

2.3.4. Истраживање примене статичке анализе кôда у индустрији

У склопу овог истраживања циљ је био истражити утицај старости и искуства запослених на мишљење о алатима за статичку анализу кôда, као и утицај алата на процесе имплементације кôда.

Ово истраживање служило је као показатељ степена употребе и примене статичке анализе кôда и алата за статичку анализу кôда у индустрији.

Истраживање је спроведено путем упитника који је обухватао три секције:

1. демографске податке,
2. употребу алата за статичку анализу кôда и
3. оцену алата.

Анкета је била дистрибуирана међу техничким особљем у области софтверског инжењерства у Новом Саду и Београду. Обрада података извршена је користећи алат *JASP (Jeffreys's Amazing Statistics Program)* [92], а за добијање резултата коришћени су Пирсонови тестови.

Анализа је показала значајне разлике у перцепцијама и учесталости коришћења алата за статичку анализу кôда, у зависности од старости и искуства софтвер инжењера. Инжењери који по старосном добу и искуству спадају у старије обично су повезани са строжим оценама ефикасности алата. Такође, показано је и да је учесталост коришћења алата већа када је њихова употреба обавезан део процеса развоја.

Резултати истраживања указују на то да искуснији софтвер инжењери имају тенденцију да буду строжи у оцени алата за статичку анализу кôда, што може бити резултат

већих очекивања и бољег разумевања потенцијалних ограничења алата. С друге стране, интеграција алата у свакодневне процесе развоја значајно повећава њихову учесталост коришћења, што је показало позитиван утицај на квалитет кôда и смањење времена потребног за одржавање кôда.

Табела 6 приказује просечне оцене које су испитаници давали за различите скупове карактеристика алата за статичку анализу кôда, разложене по категоријама старости и искуства учесника. Ове оцене одражавају варијације у перцепцији искуства софтвер инжењера. Видљиво је да млађи софтвер инжењери (енгл. *junior*) имају тенденцију да дају више оцене, што може указивати на мање искуства и нижа очекивања у поређењу са искуснијим колегама (енгл. *medior*) и најискуснијим колегама (енгл. *senior*), према овој категоризацији.

Табела 6 – Просечне оцене алата за статичку анализу кôда по искуственим групама софтвер инжењера

Скуп карактеристика алата за статичку анализу кôда	***Junior***	***Medior***	***Senior***
Задовољство перформансама алата	4.07	3.61	3.61
Интуитивност коришћења алата	3.93	3.77	3.69
Јасноћа и тачност информација	3.86	3.08	3.54
Способност писања квалитетног кôда	4.14	3.85	3.62

Истраживање потврђује утицај старости софтвер инжењера на перцепције и коришћење алата за статичку анализу кôда. Конкретно, показано је да су искуснији програмери склонији критичнијем приступу у оцењивању алата, док су млађи програмери отворенији и имају позитивније виђење истих. Учесталост коришћења алата за статичку анализу кôда је директно повезана са њиховом обавезном употребом у процесима развоја, указујући на њихову централну улогу у савременим методологијама развоја софтвера.

Истраживање је такође показало da испитаници у великој мери користе алате за статичку анализу кôда, чиме се потврђује њихова вредност у пракси развоја софтвера. Већина испитаника користи ове алате за идентификацију проблема са кôдом који могу утицати на квалитет и сигурност производа. Ово указује на снажну свест о значају квалитетног кодирања и проактивном управљању потенцијалним ризицима у развоју софтвера.

2.4 Истраживачка питања, истраживачки модели и хипотезе

У склопу овог поглавља, најпре ће бити дефинисан општи циљ истраживања:

Циљ истраживања јесте да се утврде кључни фактори који утичу на квалитет кôда студентских пројеката и да се истражи како квалитет кôда утиче на академски успех студената. Такође, циљ је и креирати модел за процес оцењивања софтверских пројеката и задатака у области софтверског инжењерства заснован на статичкој анализи кôда.

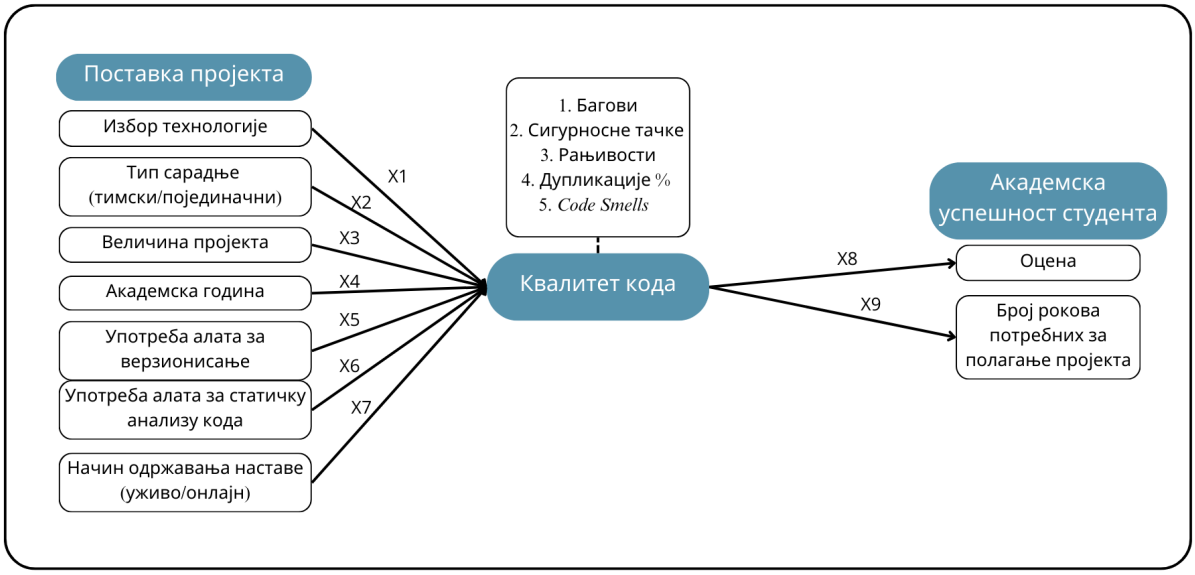
На основу представљених прегледа стања у области која се тичу два дела истраживања докторске дисертације, аутор дефинише два засебна истраживачка модела и одговарајуће хипотезе, како би се одговорило на постављена истраживачка питања и остварио постављен циљ истраживања. Питања у склопу првог истраживачког модела и припадајуће хипотезе дефинисане су на основу примарних студија издвојених из прегледа стања у области описаних у оквиру поглавља 2.3.1. и 2.3.2. Питање у склопу другог истраживачког модела и припадајуће хипотезе дефинисане су на основу примарних студија описаних у оквиру поглавља 2.3.3.

Прва два истраживачка питања гласе:

ИП1: У којој мери фактори поставке пројекта утичу на квалитет кôда студентских пројеката?

ИП2: У којој мери квалитет кôда студентских пројеката утиче на академску успешност студената?

Слика 9 представља истраживачки модел 1 (ИМ1) са хипотезама о везама између кључних чинилаца истраживања.



Слика 9 – Истраживачки модел 1 (ИМ1)

У наредном тексту описани су кључни чиниоци истраживања и теоријске основе за дефинисање њихових веза.

Избор технологије за развој софтвера може значајно утицати на учесталост различитих врста неправилности у кôду, што значајно утиче на квалитет софтвера [71]. Наведена студија указује да функционални језици доводе до бољег квалитета кôда у поређењу са процедуралним језицима. Слично томе, друга студија [80] истиче повећану комплексност језика. Додатно, студија [81] директно упоређује програмирање у језицима *C* и *C++* и показује да употреба *C++* смањује комплексност, као и број других неправилности. На основу описаних студија, у склопу истраживања потребно је избор технологије поставити као потенцијалну детерминанту квалитета кôда, претпостављајући да избор технологије може утицати на исход квалитета кôда у студентским пројектима.

Метрика која ће бити постављена у истраживању као одредница квалитета кôда јесте начин сарадње, што обухвата могућности да су пројекти организовани као индивидуални или тимски радови. У истраживање потребно је укључити оба типа организације пројекта како би се утврдило да ли је посматрана метрика одредница квалитета кôда. Претходне студије [44], [69] наглашавају утицај тимских пројеката на квалитета кôда.

Индикатори у индустрији сугеришу да квалитет кôда опада како величина пројекта расте [75]. Број линија кôда развијених у пројекту узима се као индикатор величине пројекта у овом истраживању. Кроз резултате ове анализе, циљ нам је да покажемо да ли студенти праве веће грешке када раде на већим пројектима, у поређењу са резултатима анализе квалитета кôда на мањим пројектима. Ово има за циљ да одговори на то да ли је величина пројекта значајна детерминанта квалитета кôда у образовању у области софтверског инжењерства.

У неким студијама је показано да искуство софтвер инжењера у индустрији игра значајну улогу у квалитету кôда који генеришу [76], [77]. Ове студије наводе различите метрике за мерење њиховог искуства. У овом истраживању, академска година у којој студент ради на одређеном пројекту постављена је као индикатор искуства студента. Ова мера искуства одражава искуство које студенти стичу у разумевању развоја софтвера.

Употреба алата за контролу верзија сматра се једном од могућих детерминанти квалитета кôда. Истраживање [78] наглашава значај коришћења алата за контролу верзија и спровођење процеса рецензије кôда између студената. Студија такође показује да употреба алата доприноси разумевању и примени стандарда кодирања, идентификацији неправилности и побољшању вештина кодирања међу студентима. Даљим истраживањем потребно је обухватити студентске пројекте који су користили алате за контролу верзија и оне који нису. На овај начин, овај рад има за циљ да покаже да ли и у којој мери употреба алата за контролу верзија доприноси вишем квалитету кôда.

Студије [65], [66] дискутују резултате употребе алата за статичку анализу кôда у процесу образовања. Даљим истраживањем неопходно је обухватити студентске пројекте унутар курсева где је употреба алата за статичку анализу била обавезна, као и пројекте развијене без алата. Ово има за циљ, не само да покаже да ли употреба алата постиже виши квалитет кôда у софтверским пројектима, већ и да пружи јаснију слику о способностима студената да рукују алатима за статичку анализу, интерпретирају резултате и исправљају детектоване неправилности.

Узроковано пандемијом *Covid-19*, настава је углавном одржавана онлајн током једног семестра академске 2019/2020 године, као и током целе академске 2020/2021 године. Студентске пројекте из ових година потребно је даље истражити како би се показало какви су резултати које студенти постижу у погледу квалитета кôда у онлајн режиму наставе. Након академских година погођених пандемијом, настава се вратила на традиционални начин, тј. одржавала се у просторијама факултета. Други део пројеката које је потребно истражити укључује пројекте који су изведени када је настава факултета вођена на традиционалан начин. Бројне студије, као што је [72], испитују утицај онлајн наставе на резултате које студенти постижу. Међутим, још увек не постоје студије које испитују утицај овог начина наставе на квалитет кôда који студенти генеришу.

Према стандарду квалитета кôда софтверског производа *ISO/IEC* 25010:2023 [35], дефинисано је девет категорија квалитета кôда:

- функционална погодност,
- ефикасност перформанси,
- компатибилност,
- интерактивна способност,
- поузданост,
- сигурност,
- одрживост,
- флексибилност и
- безбедност.

Алат који ће бити коришћен у истраживању, *SonarQube*, бави се категоријама поузданости, одрживости и сигурности из стандарда *ISO/IEC* 25010:2023 унутар својих основних правила квалитета кôда [93]. Остале категорије из [35] су мање наглашене јер

често захтевају специфично знање и контекст који превазилази могућности статичке анализе кôда.

Поред описаних категорија, *SonarQube* такође укључује категорију сложености изворног кôда и наглашава важност описаних метрика у приказивању нивоа квалитета изворног кôда у својој документацији. Категорија метрика које квантитативно описују квалитет кôда и које ће бити коришћене у даљем истраживању су [93]:

- Дупликација кôда повећава софтверску сложеност и отежава одржавање, што директно утиче на смањење квалитета кôда. Уклањањем дупликата, тимови за развој могу значајно побољшати поузданост и ефикасност софтвера, олакшавајући ажурирања и смањујући потенцијалне грешке [94].
- Сигурносне тачке и рањивости играју кључну улогу у процени квалитета кôда, јер њихово присуство може озбиљно угрозити поузданост и одрживост софтвера. Откривање и исправљање рањивости не само да повећава сигурност софтвера, већ доприноси и свеукупном побољшању квалитета кôда, чинећи га отпорнијим на нападе и лакшим за одржавање. Интеграција мерења сигурносних рањивости као метрике квалитета кôда омогућавају организацијама да развијају робустније и сигурније софтверске производе, чиме се подиже укупни стандард квалитета у софтверској индустрији [95].
- Грешке су директни показатељи квалитета процеса развоја софтвера и неопходне су за одржавање функционалности софтвера [1].
- *Code smells* указују на критичне мане у дизајну, имплементацији или процесима одржавања који би потенцијално могли деградирати квалитет софтвера, истичући потребу за континуираним истраживањем и напредним методама детекције како би се осигурао развој софтвера високог квалитета [96], [97].

SonarQube квантитативно представља све ове метрике у извештајима које генерише и ти резултати ће бити коришћени у истраживању. Квантитативно мерење се, у овом случају, односи на број забележених нерегуларности из одређене категорије, пронађених унутар изворног кôда софтверских пројеката.

Поред броја забележених нерегуларности, овај алат такође генерише додатне информације о специфичним нерегуларностима, као што су тежина, означавајући сваку нерегуларност неким од следећих нивоа опасности: блокатор, који означава највиши ниво опасности, затим, критично, већа нерегуларност, мања нерегуларност и слично. Додатни атрибути који описују нерегуларности неће бити узети у обзир током анализе у даљем истраживању.

Поред метрика које се користе као детерминанте квалитета кôда, истраживање такође успоставља метрике академске успешности, за које је потребно испитати корелацију са квалитетом кôда студентских пројеката. Метрике академске успешности које ће бити постављене су оцене које студенти постижу на курсу у оквиру којег је анализирани пројекат реализован, као и број покушаја који је потребан студенту како би положио пројекат.

У истраживањима [73], [84], оцене се сматрају индикатором учинка на свим нивоима образовања. Као што је већ описано, прва метрика академске успешности студента је оцена коју студент постиже на испиту. Могуће вредности за метрику оцене крећу се од 6 до 10, као и ознака 'НП' (Није Положио), која се додељује студентима који још нису завршили курс. Циљ праћења ове метрике је анализа односа између резултата квалитета кôда и оцене коју студент постиже на курсу.

У склопу првог истраживачког модела, као део испитивања истраживачких питања ИП1 и ИП2, формулисано је 9 хипотеза:

Х1: Избор технологије има утицај на квалитет кôда софтверског пројекта.

Х2: Тип сарадње на пројекту има утицај на на квалитет кôда софтверског пројекта.

Х3: Величина пројекта утиче на квалитет кôда.

Х4: Академска година студента позитивно утиче на квалитет развијеног кôда.

Х5: Употреба алата за верзионисање позитивно утиче на квалитет кôда.

Х6: Употреба алата за статичку анализу кôда позитивно утиче на квалитет кôда.

Х7: Начин одржавања наставе утиче на квалитет кôда.

Х8: Квалитет кôда је у позитивној корелацији са оценом студента.

Х9: Квалитет кôда је у позитивној корелацији са бројем рокова потребним за завршетак пројекта.

Треће истраживачко питање гласи:

ИП3: Како наставници оцењују важност и потребу за алатима за проверу плагијаризма и квалитета кôда у току прегледања софтверских пројеката?

Слика 10 представља истраживачки модел 2 (ИМ2) са хипотезама о везама између кључних чиниоца истраживања.

Слика 10 – Истраживачки модел 2 (ИМ2)

У наредном тексту описани су кључни чиниоци истраживања и теоријске основе за дефинисање њихових веза.

У циљу свеобухватног истраживања квалитета кôда у студентским пројектима, укључена је варијабла која се односи на плагијаризам у изворном кôду [85], [87]. Ова варијабла омогућава анализу учесталости плагијаризма у студентским радовима, као и идентификацију најчешћих облика и метода за копирање кôда које користе студенти. Поред тога, истражује се и које алате и методе наставници примењују како би детектовали плагијаризам. Утврђивање плагијаризма кључно је за одржавање академске честитости и квалитета образовања, те се стога користе напредни алати попут алгоритама за детекцију сличности и софтверских алата специјализованих за анализу кôда. Овај аспект истраживања директно доприноси разумевању начина на који се академски интегритет може очувати и побољшати у образовном процесу.

Додатна варијабла која се разматра у истраживању односи се на примену алата за аутоматско оцењивање у процесу оцењивања студентских пројеката. Алати, који се заснивају на принципима статичке анализе кôда, омогућавају брзо и објективно оцењивање софтверских пројеката, чиме се убрзава процес оцењивања и повећава његова транспарентност. Анализа обухвата како и у којој мери наставници користе ове алате, као и ефикасност алата у процени квалитета кôда у односу на традиционалне методе оцењивања. Коришћењем алата, наставници могу детаљно анализирати квалитет кôда студената, идентификовати честе грешке и слабости и пружити прецизне повратне информације, што потенцијално води до побољшања образовних исхода и развоја вештина кодирања међу студентима.

У склопу другог истраживачког модела који би требало да одговори на ИП3, дефинисане су хипотезе:

X10: Наставници, који сматрају проблем плагијата значајним, показују већу спремност за коришћење алата за проверу плагијата и процену квалитета кôда.

X11: Наставници, који сматрају преглед великог броја студентских пројеката без алата изазовним, показују већу спремност за коришћење алата за проверу плагијата и процену квалитета кôда.

X12: Наставници, који користе алате за проверу плагијата, сматрају да је проблем плагијата значајнији у односу на наставнике који не користе те алате.

X13: Већина наставника сматра да је употреба алата за аутоматско оцењивање и проверу плагијата валидан и адекватан метод у процесу оцењивања студентских софтверских пројеката.

У наредном поглављу биће описана методологија прикупљања и обраде података за оба истраживачка модела.

3. Методолошки аспекти истраживања

Након дефинисања дизајна истраживања кроз формулисане истраживачке проблеме, утврђивање потребе за истраживањем и прегледа кључних чиниоца истраживања, потребно је преусмерити пажњу на методолошке аспекте истраживања. Методолошки аспекти истраживања подразумевају опис методологије спровођења истраживања, дефинисање инструмената за прикупљање података и представљање плана обраде тих података, који су условљени самим дизајном истраживања [98].

С обзиром на то да докторска дисертација садржи два истраживачка модела, који захтевају две различите методологије прикупљања и обраде података, ово поглавље се даље дели на две целине.

3.1 Методологија обраде података истраживачког модела 1 (ИМ1)

Истраживачки модел 1 (ИМ1) се односи на испитивање везе између фактора поставке пројекта и квалитета кôда мереног употребом алата за статичку анализу кôда. Поред тога, испитивана је и веза између квалитета кôда и академског успеха студената са циљем провере да ли постоји корелација између описаних група варијабли. Прикупљање података за овај истраживачки модел подразумева прикупљање и избор пројеката за спровођење истраживања, прикупљање информација о поставци пројекта, анализу пројеката изабраним алатом за статичку анализу кôда, синтеза резултата добијених из извештаја генерисаних од стране алата за статичку анализу кôда и прикупљање података о академском успеху студената чији су пројекти укључени у истраживање.

Након прикупљања података, неопходно је обрадити прикупљене податке и припремити их за статистичку анализу. Као финални корак, потребно је спровести статистичку анализу над подацима и дискутовати добијене резултате у односу на постављен истраживачки модел.

3.1.1. Прикупљање података

За потребе овог истраживања, прикупљање података је извршено кроз анализу преко 500 студентских пројеката развијаних у оквиру различитих предмета на Факултету техничких наука, Универзитета у Новом Саду. Пројекти су обухватали софтверске апликације и системе који су студенти развијали током студија, користећи различите програмске језике и технолошке оквире, као што су *.NET, Java, Spring, Angular*, и други. Овај широк спектар технологија омогућио је свеобухватну анализу различитих врста софтвера, што је допринело репрезентативности узорка.

Пројекти, који су обухваћени истраживањем, развијани су током неколико академских година, што је омогућило да се у истраживање укључе пројекти из различитих фаза студија. Узорковани су пројекти из прве, друге, треће и четврте године студија. На тај начин укључени су пројекти студената са различитим нивоима искуства и вештина у програмирању.

Пројекти су такође укључивали различите модалитете извођења наставе. Ово се односи на оне пројекте који су рађени у условима традиционалног извођења наставе (уживо) и на оне који су реализовани у условима онлајн наставе, што је постигнуто укључивањем пројеката током периода пандемије *COVID-19*, када је значајан део наставе реализован у онлајн режиму.

Узорак је укључивао студентске пројекте са различитим обимом, у смислу броја линија кôда и сложености функција које су пројекти имплементирали. Величина пројеката се кретала од малих, са неколико стотина линија кôда, до великих пројеката са хиљадама линија кôда. Наведени пројекти су развијани самостално или у тимовима, чиме је омогућено испитивање утицаја тимског и индивидуалног рада на квалитет кôда.

Прикупљање метрика квалитета кôда реализовано је коришћењем алата за статичку анализу кôда – *SonarQube* [30]. Детаљан опис процедуре избора алата за статичку анализу кôда биће описан у поглављу 3.1.5. Овај алат омогућава дубинску анализу изворног кôда без његовог извршавања и генерише различите метрике које процењују квалитет кôда у смислу:

- Грешака**:** Проблеми у кôду који могу довести до нефункционалности или грешака при извршавању.
- Дупликација: Проценат кôда који је дуплиран у различитим деловима пројекта, што повећава сложеност и смањује одрживост.
- Рањивости: Потенцијалне сигурносне рупе које могу угрозити безбедност апликације.
- *Code smells*: Структурални проблеми у кôду који указују на лоше праксе у развоју и могу довести до будућих проблема у одржавању.

SonarQube омогућава анализу кôда у више програмских језика (*Java, C#, JavaScript, HTML/CSS*), што је било кључно с обзиром на различите технологије које су студенти користили.

Поред метрика квалитета кôда, прикупљени су и подаци о академској успешности студената. За сваки пројекат узете су у обзир оцене које су студенти добили на курсу, као и број покушаја који су били потребни за успешну одбрану и полагање пројекта. Овај скуп података омогућио је истраживачима да испитају корелацију између квалитета кôда и академског успеха студената.

Приликом избора пројеката који ће бити укључени у истраживање постављена су правила избора. Ова правила омогућила су да пројекти из сваке категорије буду садржани у узорку пројеката који су укључени у истраживање.

Пример прикупљених и необрађених података у склопу истраживања приказани су у оквиру прилога А.

3.1.2. Обрада података

Фаза обраде података је кључна за припрему скупа података за даљу анализу и статистичку обраду. Обрада података обухватила је неколико корака:

1. чишћење података,
2. кодирање променљивих,
3. стандардизацију континуалних променљивих и
4. креирање нових променљивих ради додатне анализе.

Коришћене метрике описане у оквиру кључних чинилаца истраживачког модела, надаље биће навођене у раду према скраћеним називима приказаним у табели 7.

Табела 7 – Скраћени називи метрика из ИМ1

Метрика	Скраћени назив
Избор технологије	ИТ
Тип сарадње	ТС
Величина пројекта	ВП
Академска година	АГ
Употреба алата за верзионисање	УАВ
Употреба алата за статичку анализу кôда	УСАК
Начин одржавања наставе	НОН
Квалитет кôда	КК
Оцена студента	ОС
Број рокова потребних за полагање пројекта	БРП

На почетку је извршено чишћење скупова података како би се уклониле некоректне или недостајуће вредности. С обзиром на то да је прикупљање података реализовано из различитих извора (*SonarQube* за метрике квалитета кôда и универзитетска база за оцене и покушаје завршетка пројеката), било је неопходно да се обезбеди конзистентност података.

- Недостајуће вредности: Пројекти са недостајућим вредностима за кључне метрике (нпр. багови, дупликације, рањивости) или недостајућим оценама су елиминисани из анализе, како би се обезбедило да сви подаци буду комплетни.
- Дуплиране вредности: У случају да су постојале дуплиране вредности за исте пројекте, задржане су само оне вредности које су биле најкомплетније и најновије, како би се избегла погрешна анализа.

Могуће вредности кључних чиниоца у склопу ИМ1 приказане су у табели 8. Метрика квалитета кôда није приказана у оквиру табеле 7 и биће детаљније описана у поглављу 3.1.3.

Табела 8– Приказ могућих вредности сваке варијабле

Метрика	Могуће вредности
ИТ	1. *.NET* 2. *Spring* и *Angular* 3. *Java* 4. *.NET - WPF* 5. *.NET* и *Angular* 6. *HTML, CSS,* и *JS*
ТС	Тимски/појединачно
ВП	Број линија кôда 0 <
АГ	I, II, III, IV
УАВ	Коришћено/није коришћено
УСАК	Коришћено/није коришћено
НОН	Онлајн/уживо
ОС	НП, 6, 7, 8, 9, 10
БРП	НП, 0 <

Након чишћења података, следећи корак је био кодирање номиналних и бинарних променљивих у одговарајући формат како би биле погодне за статистичку анализу:

- Бинарне променљиве: Променљиве које представљају бинарне одлуке, као што су коришћење алата за статичку анализу кôда (УСАК), коришћење алата за верзионисање (УАВ), модалитет наставе (НОН – онлајн или уживо) и тип сарадње (ТС – тимски или индивидуални рад), кодиране су на следећи начин:
    - 1 за коришћење алата или учешће у одређеном модалитету (нпр. ако је студент користио алат за статичку анализу кôда, вредност је 1).
    - 0 за некоришћење алата или одсуство из модалитета.
- Номиналне променљиве: Номиналне променљиве које представљају избор технологије (ИТ) су кодиране на основу учесталости појављивања категорија у бази података. То значи да су технологије попут *.NET, Java, HTML/CSS/JS* и *Spring* добиле одређене нумеричке вредности у складу са учесталошћу њихове употребе у пројектима.
    - На пример, ако је највише пројеката рађено у *.NET* технологији, ова категорија је добила вредност 1, *Java* је добила вредност 2 итд. Ово је урађено како би се избегло додатно повећање димензионалности података кроз *dummy* кодирање (кодирање које не генерише никакву функционалност) и како би се задржала релативна учесталост коришћених технологија.

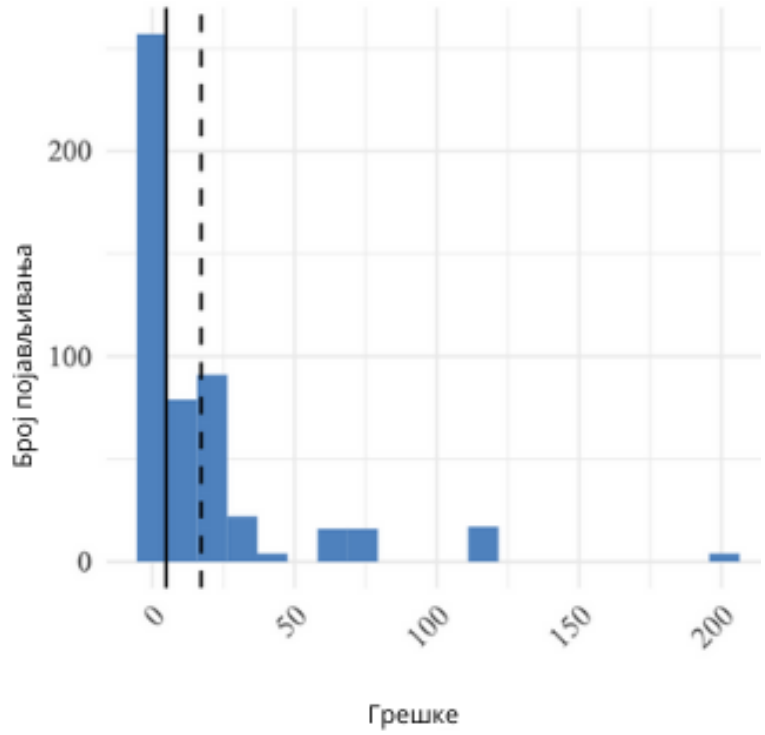
Континуалне променљиве које су представљале метрике квалитета кôда (нпр. број багова, проценат дуплирања, број *code smells*, рањивости и безбедносни пропусти) биле су на различитим скалама, што је могло утицати на резултате анализе. Због тога је извршена стандардизација променљивих, што је укључивало следеће кораке:

- Стандардизација се састојала у претварању вредности сваке променљиве на скалу са средњом вредношћу од 0 и варијансом од 1.

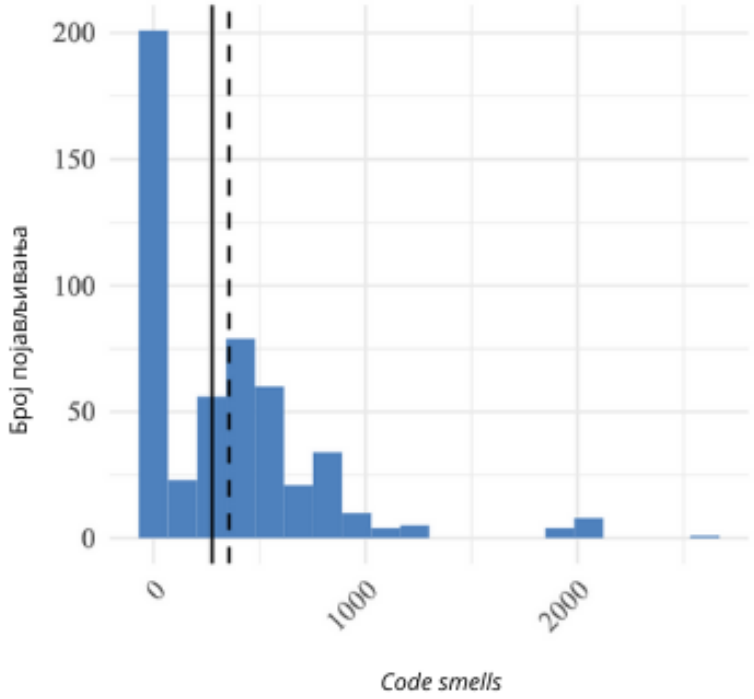
Ово је омогућило да се сви подаци обрађују на истој скали, чиме се избегава да једна променљива доминира резултатима само због веће величине својих вредности.

- Дистрибуција података: У овом кораку је такође проверена дистрибуција свих континуалних променљивих како би се уочила било каква права или закошеност. За променљиве које су показале знатно одступање од нормалне дистрибуције извршена је трансформација података како би се обезбедила симетричнија дистрибуција.

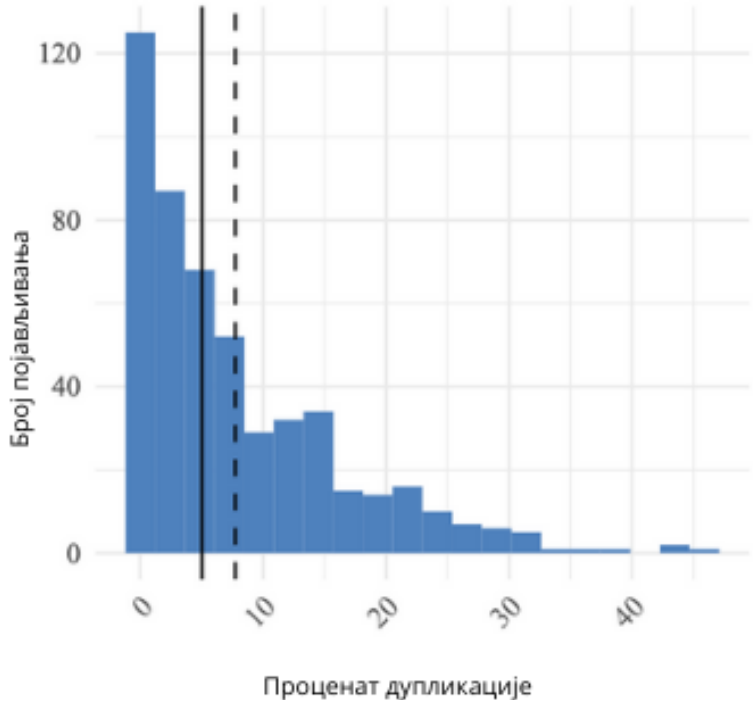
Приказ дескриптивне статистике прикупљених и обрађених података за континуалне метрике се налази на сликама 11 – 15. Слике презентују заступљеност и расподелу појављивања сваке варијабле у подацима, као и ознаке медијане (пуна линија) и модус (испрекидана линија) над бележеним подацима.



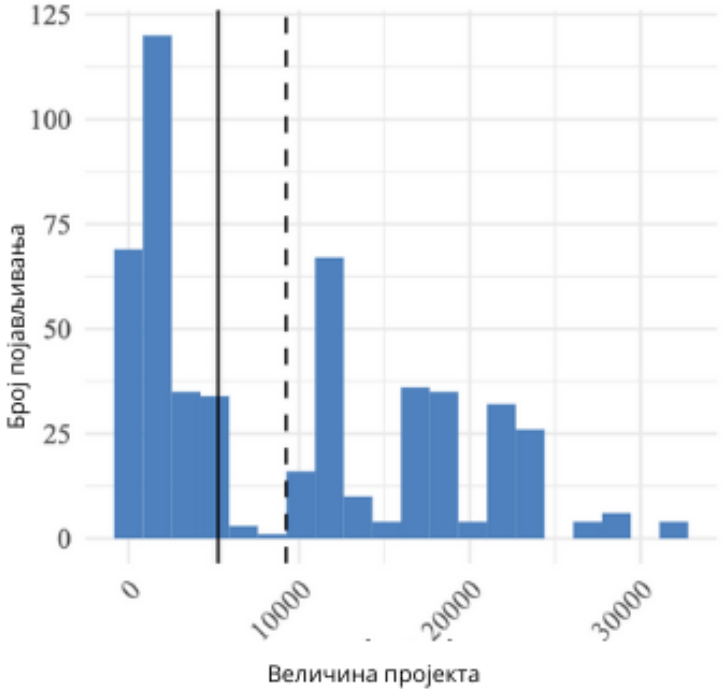
Слика 11 – Расподела броја појављивања метрике багова (грешака) у подацима



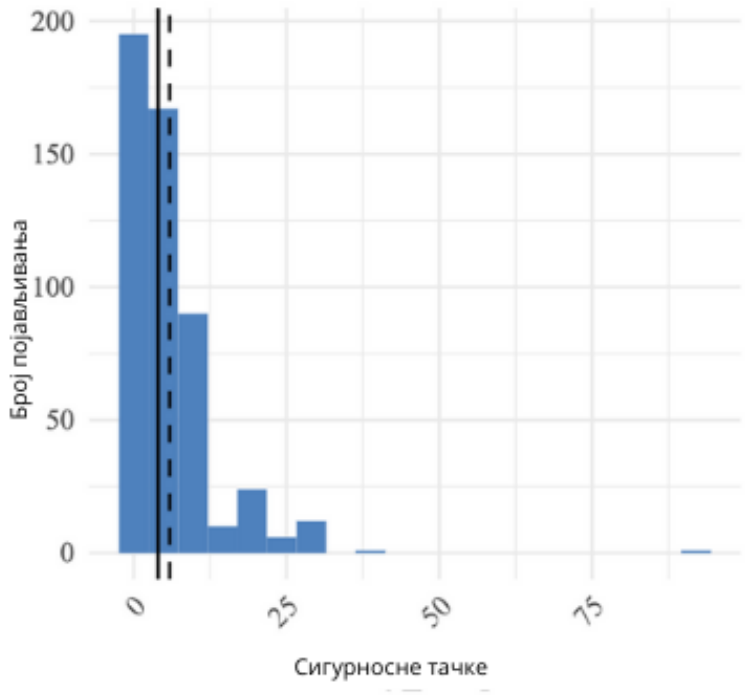
Слика 12 – Расподела броја појављивања метрике *code smells* у подацима



Слика 13 – Расподела броја појављивања метрике дупликације (%) у подацима



Слика 14 – Расподела броја појављивања метрике величина пројекта у подацима

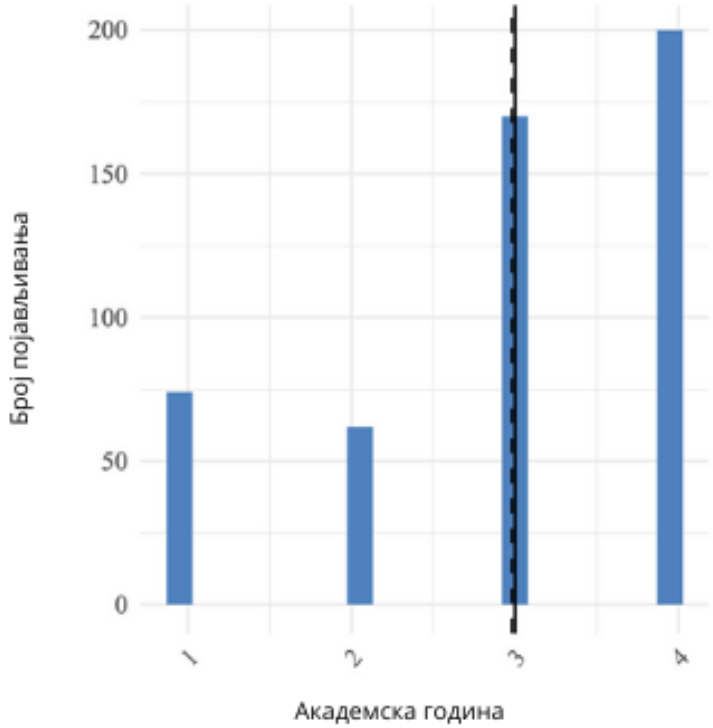


Слика 15 – Расподела броја појављивања метрике сигурносних тачки у подацима

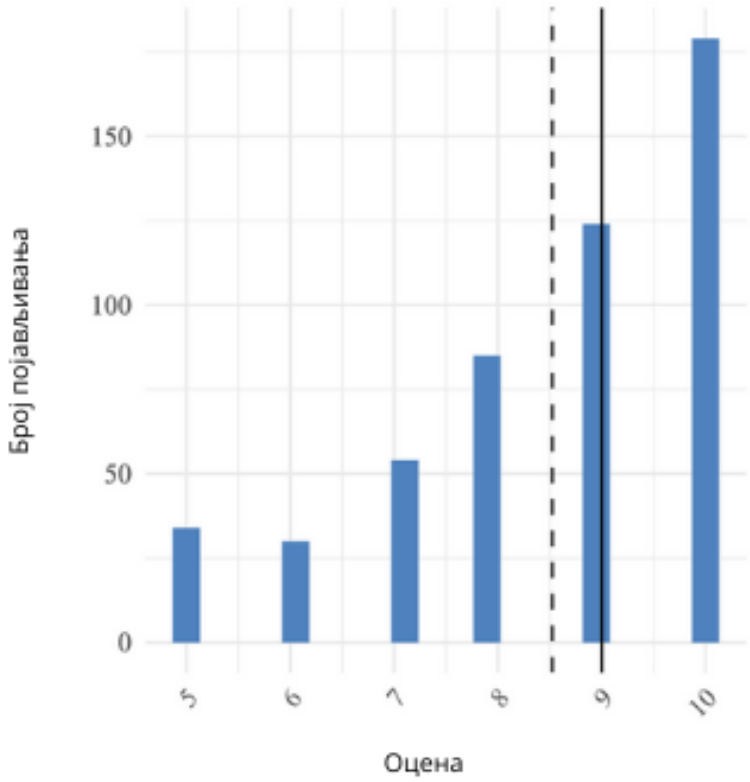
Да би се утврдило да ли постоји висока корелација између различитих независних променљивих, извршена је анализа корелације. За потребе анализе је коришћен Кендалов коефицијент корелације. Кендалов коефицијент корелације, познат и као Кендалов τ (*Tau*), служи за мерење јачине и правца везе између две рангиране променљиве. У случају да је било променљивих које показују висок степен мултиколинеарности, примењена је техника *Variance Inflation Factor* (*VIF*), која показује до које мере мултиколинеарност утиче на варијансу процене регресионих коефицијената. Вредности *VIF* преко 5 сматране су индикативним за високу колинеарност, што је захтевало елиминацију или трансформацију неких променљивих.

У оквиру обраде података, посебна пажња је посвећена пречишћавању резултата које је генерисао алат за статичку анализу кôда. Било је важно идентификовати и потенцијално уклонити лажне позитивне резултате које је алат могао генерисати. Лажни позитивни резултати у контексту статичке анализе кôда односе се на откривање проблема у кôду који не представљају стварне грешке, што може утицати на тачност закључака о квалитету кôда [14].

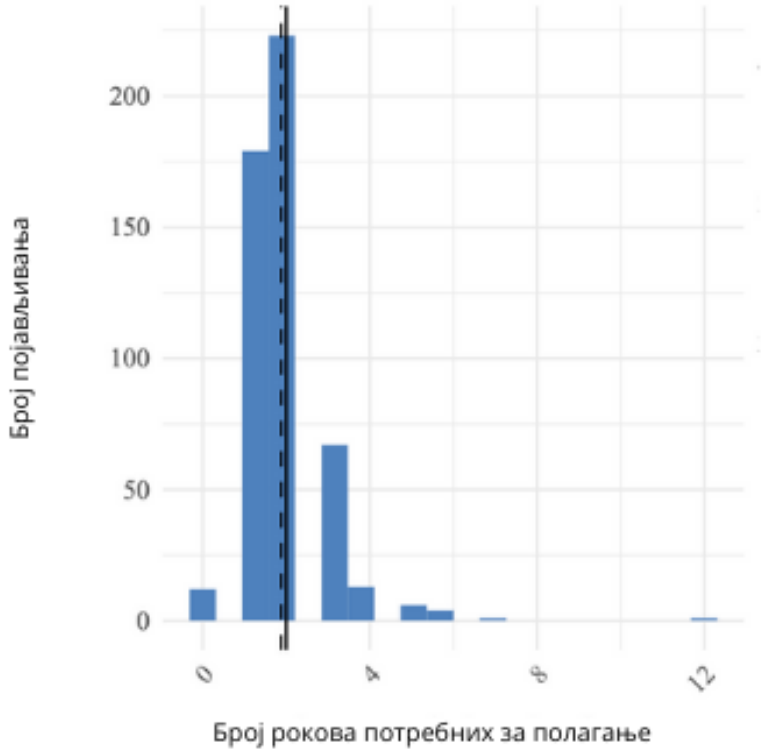
Приказ дескриптивне статистике прикупљених и обрађених података за ординалне метрике приказано је на сликама 16 – 19. На сликама је представљена заступљеност и расподела појављивања сваке варијабле у подацима, као и ознаке медијане (пуна линија) и модус (испрекидана линија) над бележеним подацима.



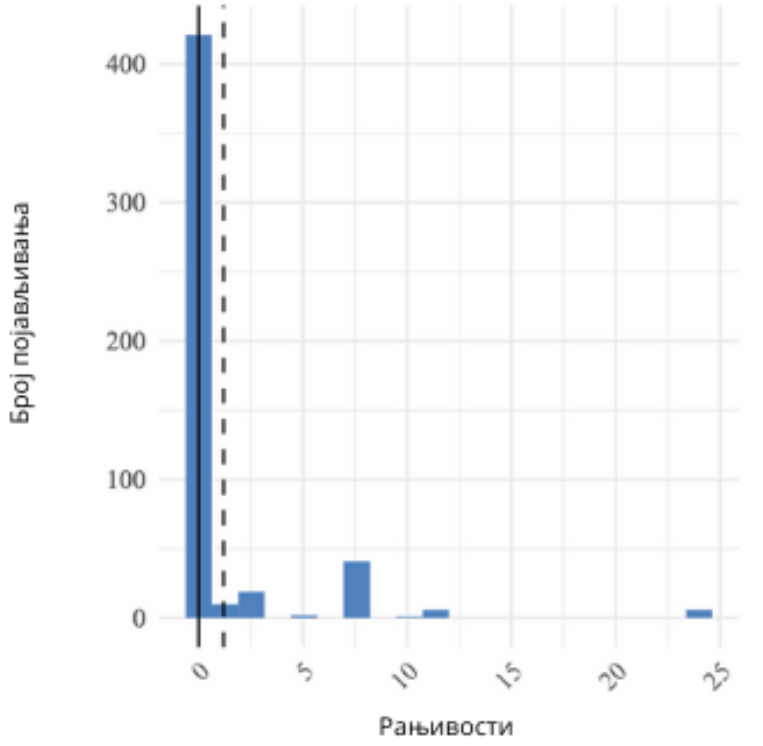
Слика 16 – Расподела броја појављивања метрике академске године у подацима



Слика 17 – Расподела броја појављивања метрике оцена у подацима



Слика 18 – Расподела броја појављивања метрике времена потребног за полагање пројекта у подацима



Слика 19 – Расподела броја појављивања метрике рањивости у подацима

Ова фаза обраде је обезбедила припремљеност свих података тако да могу бити искоришћени за дубљу статистичку анализу. Уклоњене су недостајуће и дуплиране вредности, а све променљиве су стандардизоване и кодиране ради лакше интерпретације и анализе.

3.1.3. Креирање нове променљиве – квалитет кôда (КК)

За потребе овог истраживања креирана је додатна зависна променљива под називом квалитет кôда (КК) како би се омогућила свеобухватна анализа утицаја различитих фактора на укупан квалитет софтверских пројеката. Ова променљива комбинује више метрика квалитета кôда (као што су већ описане грешке, дупликације, безбедносни пропусти, *code smells*, и рањивости) у једну променљиву која ће представљати свеукупни квалитет кôда.

Иако је *SonarQube* пружио велики број појединачних метрика које се односе на квалитет кôда, свака од њих одвојено не даје свеобухватну слику о укупном квалитету пројекта. На пример, пројекат може имати велики број грешака, али мало дупликација или рањивости, што ствара разнолике резултате за различите метрике. У циљу добијања јасније слике о укупном квалитету кôда, било је неопходно интегрисати ове метрике у једну свеобухватну променљиву.

Креирањем КК променљиве омогућено је да се испита квалитет кôда у односу на различите факторе као што су технологија која је коришћена, величина пројекта, начин сарадње, коришћење алата за контролу верзија и други описани фактори.

Нова КК променљива конструисана је као бинарна променљива са могућим вредностима 0 и 1, где:

- 1 означава висок квалитет кôда, односно кôд који задовољава свеобухватни критеријум квалитета.
- 0 означава низак квалитет кôда, где пројекат није постигао минималне стандарде квалитета у дефинисаним метрикама.

Формула за израчунавање КК укључивала је пет основних метрика квалитета кôда (багови, дупликације, рањивости, *code smells* и безбедносни пропусти), при чему је свакој метрици додељена једнака тежина у финалном израчунавању КК. Све метрике су прво стандардизоване како би биле на истој скали, а затим су агрегиране у складу са следећом формулом:

$$\text{КК} = \text{Б} \times W\text{Б} + \text{Д} \times W\text{Д} + \text{Р} \times W\text{Р} + CS \times WCS + \text{БП} \times W\text{БП}$$

Где су:

- Б: број грешака у кôду,
- Д: проценат дупликација у кôду,
- Р: број рањивости,
- *CS*: број *code smells*,
- БП: број безбедносних пропуста.

Тежина сваке метрике (W) постављена је на 0.2, што значи да свака метрика учествује са једнаком тежином у коначној процени квалитета кôда. Ова формула обезбеђује уравнотежену процену квалитета, при чему свака метрика доприноси финалном резултату.

Праг за класификацију пројекта као висококвалитетног (КК = 1) постављен је на 0.5. То значи да ако укупна агрегирана вредност КК променљиве (на основу тежина и вредности метрика) премаши 0.5, пројекат је класификован као висококвалитетан. Ако је резултат испод 0.5, пројекат је означен ниском квалитетношћу.

На пример, уколико пројекат има релативно низак број грешака, али велики број *code smells*, укупна КК вредност може бити ближа 0.5, али ако друге метрике попут дупликација и безбедносних пропуста имају добре резултате, пројекат би могао бити класификован као висококвалитетан.

Избор бинарне класификације је оправдан чињеницом да је ова метода једноставна за примену и анализу, посебно у контексту биномне логистичке регресије која је коришћена у наредној фази анализе. Бинарна КК променљива омогућава лакше тумачење утицаја различитих независних променљивих на квалитет кôда, јер је циљ био испитати који фактори (као што су величина пројекта, коришћење алата за статичку анализу, итд.) имају највећи утицај на постизање висококвалитетног кôда.

Иако је КК променљива омогућила свеобухватну анализу квалитета кôда, постоје и одређена ограничења у њеној примени:

- Лажни позитивни резултати: Могућност да *SonarQube* генерише лажне позитивне резултате могла је утицати на крајњу вредност КК променљиве. Овај фактор је делимично ублажен пречишћавањем података и проверама валидности резултата.
- Једнаке тежине метрика: Иако су све метрике добиле једнаке тежине у финалној формули, може се аргументовати да неке метрике имају већи утицај на квалитет кôда од других (на пример, безбедносни пропусти могу бити важнији од дупликација у одређеним случајевима).

Овај процес је обезбедио интегрисану и једноставну меру за свеобухватан квалитет студентских пројеката. Креирање ове променљиве омогућило је дубље разумевање утицаја различитих фактора на квалитет кôда и помогло у даљој анализи корелација са академским успехом студената.

3.1.4. Избор статистичких тестова и анализа

Након прикупљања и обраде података, следећи корак је била статистичка анализа која је омогућила испитивање утицаја различитих независних променљивих на новокреирану променљиву КК, као и на академске перформансе студената. У овој фази анализе коришћене су различите статистичке технике како би се открили односи између фактора као што су величина пројекта, коришћење алата за статичку анализу, начин сарадње и други, са резултатима квалитета кôда и оценама студената.

Први корак у статистичкој анализи био је испитивање корелације између независних променљивих и променљиве КК. За ову сврху коришћен је Кендалов коефицијент корелације, који мери асоцијацију између рангираних променљивих. Овај метод је изабран због:

- Способности да измери корелацију у случају када променљиве нису нормално распоређене, што је био случај са неким од континуалних променљивих у овом истраживању.
- Способности да се носи са рангираним и бинарним подацима. Кендалов коефицијент корелације (τ) израчунава вредности у распону од -1 до 1, где:
  - **1** означава савршену позитивну корелацију,
  - **-1** означава савршену негативну корелацију и
  - **0** означава одсуство корелације.

Ова анализа омогућила је идентификацију који фактори имају највећи утицај на квалитет кôда.

Главни метод који је коришћен за испитивање утицаја независних променљивих на КК променљиву била је биномна логистичка регресија. Овај метод је коришћен јер је зависна променљива КК била бинарна (0 и 1), што значи да је модел требало да испита вероватноћу да одређени пројекат има висок или низак квалитет кôда у односу на различите факторе.

Логистичка регресија омогућава:

- Израчунавање утицаја сваке независне променљиве на вероватноћу да пројекат постигне висок квалитет кôда.
- Тумачење резултата у облику коефицијената, где се сваки коефицијент може интерпретирати као промена у логаритму шанси да квалитет кôда буде висок у односу на промену независне променљиве.

Крајњи резултати логистичке регресије приказани су као односи шанси (енгл. *odds ratio*), што показује колико пута је већа вероватноћа да ће пројекат бити високог квалитета уколико је независна променљива присутна.

Да би се испитала мултиколинеарност између независних променљивих, коришћен је већ поменути *Variance Inflation Factor* (*VIF*). Овај фактор мери колико је варијанса процене коефицијената у регресионом моделу прецизна, због могуће корелације између независних променљивих.

Да би се испитала разлика у академској успешности студената у односу на квалитет кôда, коришћен је *Mann-Whitney U* тест. Овај тест је примењен како би се утврдило да ли постоје статистички значајне разлике у оценама студената (ОС) и броју покушаја за завршетак пројекта (БРП) у зависности од тога да ли је пројекат имао висок или низак квалитет кôда.

Mann-Whitney U тест је изабран јер је податке о оценама и броју покушаја било неопходно третирати као нормално нераспоређене и ординисане променљиве.

У последњем кораку анализе, испитан је синергетски ефекат између неких независних променљивих, попут величине пројекта (ВП) и избора технологије (ИТ). Кроз парцијалне графиконе показано је да одређене комбинације фактора значајно повећавају шансе за постизање висококвалитетног кôда.

Различити тестови и регресиони модели који су коришћени омогућили су дубинско разумевање начина на који различити фактори утичу на квалитет кôда. Коришћење биномне логистичке регресије, *Mann-Whitney U* теста и других метода омогућило је прецизно утврђивање односа између независних променљивих и зависне променљиве (КК), као и академске успешности студената. Добијени резултати свих примењених тестова биће приказани у поглављу 4.

3.1.5. Избор алата за статичку анализу кôда који ће бити коришћен у истраживању

У овом поглављу ће бити описан начин избора алата за статичку анализу кôда који ће бити коришћен за оцену квалитета кôда софтверских пројеката. Као први корак издвојени су алати који се најчешће користе у истраживањима (слика 6), над овако издвојеним алатима спроведена је анализа коришћењем *DESMET* [99] (*A Methodology for Evaluating Software Engineering Methods and Tools*) методологије.

DESMET је методологија за евалуацију алата или метода. Развила ју је Барбара Киченхам [100] и састоји се од девет метода евалуације погодних за различите типове алата. У *DESMET* методологији постоје квантитативне и квалитативне методе. Први корак у примени *DESMET* методологије је одабир одговарајућег метода евалуације. *DESMET* методологија има одређена ограничења када се комбинују и мешају методе и алати, нарочито ако у организацији нема контролисаног развојног процеса. Ауторка је дефинисала критеријуме на основу којих се бирају методе у оквиру *DESMET* методологије.

Анализа карактеристика је квалитативни облик евалуације који укључује субјективну процену релативне важности различитих карактеристика, као и процену колико је добро свака од карактеристика имплементирана код кандидата за алате. Циљ анализе карактеристика је да пружи основу за доношење одлуке да ли користити одређени алат у организацији. Евалуација укључује следеће области:

- погодност за сврху,
- економски аспекти,
- недостаци и
- остале предности.

Ова методологија се сматра субјективном методом евалуације. Критеријуми који би требало да се размотре приликом доношења одлуке о спровођењу анализе карактеристика укључују:

- велики број метода/алата за процену и
- кратке временске рокове за евалуацију.

Три алата која над којима ће бити примењен овај метод оцењивања су:

1. *Cppcheck* [101]: Алат за статичку анализу, најпре намењен за анализе пројеката у *C/C++* програмском језику. Обезбеђује јединствену анализу кôда ради откривања грешака и фокусира се на детекцију неодређеног понашања и опасних кодних конструкција.
2. *FindBugs* [34]: Програм који користи статичку анализу ради проналажења грешака, најпре намењен за пројекте написане у Java програмском језику. Ово је софтвер отвореног кôда.
3. *SonarQube* [30]: Отворена платформа за континуирану инспекцију квалитета кôда. Sonar обавља статичку анализу кôда и даје детаљне извештаје о грешкама, code smells и рањивостима.

Типови карактеристика који представљају димензије анализе у склопу *DESMET* методологије подељене су у четири сета: економија, лакоћа увођења, подршка циклусу прегледа кôда и управљање процесом. У следећем делу описујемо карактеристике у сваком од сетова. У наставку биће описан сваки од наведених сетова карактеристика:

1. Економичност: Овај сет се односи на економске факторе у вези са почетним трошковима алата и накнадном подршком за одржавање или надоградњу алата. У овом истраживању, највиши бодови су додељени ако није потребно почетно плаћање (*F1-SF01*) и ако алат има добро одржавање које је бесплатно, укључујући редовне исправке и једну контакт тачку за кориснике у случају потребе за подршком (*F1-SF02*).

2. Лакоћа увођења: Овај сет карактеристика фокусиран је на ниво тешкоће приликом првог подешавања и коришћења алата. Сваки алат би требало да има разумне системске захтеве (*F2-SF01*) и не захтева напредан хардвер или софтвер за функционисање. Затим је потребно да има једноставну процедуру инсталације и подешавања (*F2-SF02*) која је подржана упутством за инсталацију (*F2-SF03*) или туторијалом (*F2-SF04*); да буде што самосталнији, тј. способан да функционише као самостална апликација уз минималне захтеве за спољним технологијама (*F2-SF05*).
3. Подршка циклусу прегледа кôда: Ове карактеристике односе се на то колико добро алат подржава стандардне фазе у циклусу прегледа кôда. Прво, у стандардном циклусу прегледа, алат би требало да обезбеди преглед свих грешака у изворном кôду програма (*F3-SF01*). Такође, алат би требало да прати исправљене грешке у процесу реинжењеринга кôда (*F3-SF02*). У фази прегледа кôда, алат би требало да обезбеди информацију где се грешка налази у изворном кôду или да означи део кôда који садржи нерегуларност (*F3-SF03*). Иако алати за статичку анализу кôда не елиминишу откривене грешке у изворном кôду, већ их само означавају, алати би требало да предложе решење за отклањање уочене грешке (*F3-SF04*). Извештаји генерисани алатима за статичку анализу би требало да указују на тип откривене грешке, као и на ниво хитности њеног исправљања (*F3-SF05*). Коначно, у овом сету карактеристика, алати би требало да покажу процењено време потребно за исправљање сваке грешке у изворном кôду (*F3-SF06*).
4. Управљање процесом: Овај сет карактеристика односи се на управљање процесом статичке анализе кôда. Статичка анализа кôда може бити колаборативни процес. Стога, алат би требало да пружи могућност вишекорисничког рада на једном пројекту (*F4-SF01*). Такође, алат би требало да подржава управљање документима (*F4-SF02*), посебно када је реч о великим пројектима. Алат би требало да буде сигуран и безбедан (*F4-SF03*) и да обезбеђује ауторизацију и аутентификацију корисника. Требало би да омогући управљање улогама корисника (*F4-SF04*), на пример, корисно је назначити који корисник ће отклањати коју врсту грешака. Коначно, алат би требало да подржава више пројеката за статичку анализу кôда (*F4-SF05*).

Три елемента процеса бодовања су:

1. Бодовање сваког алата у односу на сваку карактеристику како би се добио сирови резултат.
2. Додељивање нивоа важности свакој карактеристици, који се користи као тежина (тј. мултипликатор) за конвертовање сирових резултата у пондерисане резултате за сваку карактеристику.
3. Одређивање резултата за сваки сет карактеристика и укупног резултата за сваки кандидатски алат.

Детаљно објашњење процеса бодовања описано је у [100]. Преглед коришћених сетова карактеристика приказан је у табели 9.

Табела 9 – Сетови карактеристика коришћени у анализи

Ознака	Сет карактеристика	Ознака	Подкарактеристика	Важност	Скала за оцену	Тежински фактор
F1	Економичност	*F1-SF01*	Алат захтева плаћање да би могао да се користи	*HD*	*JI1*	0.1
		F1-SF02	Одржавање	*HD*	*JI1*	
F2	Лакоћа инсталације	*F2-SF01*	Алат има системске захтеве који су разумљиви.	*M*	*JI1*	0.2
		F2-SF02	Лака инсталација и подешавање	*HD*	*JI2*	
		F2-SF03	Постоји упутство за инсталацију	*HD*	*JI1*	
		F2-SF04	Постоји туторијал	*HD*	*JI1*	
		F2-SF05	Алат је самоодржив	*HD*	*JI1*	
F3	Подршка циклусу прегледа кôда	*F3-SF01*	Дат је прегледни приказ неправилности у кôду	*D*	*JI1*	0.4
		F3-SF02	Омогућено је праћење исправљених неправилности	*HD*	*JI3*	
		F3-SF03	Приказана је локација где се налази неправилност	*HD*	*JI3*	
		F3-SF04	Предложен је начин исправљања	*HD*	*JI3*	
		F3-SF05	Идентификован је тип неправилности	*N*	*JI1*	
		F3-SF06	Приказана је естимација времена потребног за отклањање	*N*	*JI1*	
F4	Управљање процесом	*F4-SF01*	Подршка за већи број корисника	*M*	*JI1*	0.3
		F4-SF02	Управљање документима	*D*	*JI1*	
		F4-SF03	Безбедност	*D*	*JI1*	
		F4-SF04	Управљање улогама	*HD*	*JI1*	
		F4-SF05	Подршка за већи број пројеката	*M*	*JI1*	

Нивои важности подкарактеристика су [100]:

1. Обавезна карактеристика – *M*: 4 поена.
2. Веома пожељна – *HD*: 3 поена.
3. Пожељна – *D*: 2 поена.
4. Добро је да постоји – *N:* 1 поен.

Тумачења скала за оцену су [100]:

1. *J11*: Да ли је карактеристика присутна?
2. *J12*: Да ли је алат једноставан за инсталацију и подешавање?
3. *J13*: Да ли је активност подржана?

У наставку ће бити приказан преглед резултата бодовања за сваки алат. Најпре резултати за алат *Cppcheck.*

1. Економичност: Алат *Cppcheck* је бесплатан за коришћење и може се преузети са веб-сајта тима који га развија. Алат се добро одржава, редовно се ажурира (последње ажурирање било је у децембру 2023. године), и обезбеђује једну контакт тачку за кориснике у случају да им је потребна помоћ. *Cppcheck* је постигао 6 од 6 бодова у овом сету карактеристика.
2. Економичност: *Cppcheck* се може преузети са сајта развојног тима и инсталирати локално. Има делимично сложен процес подешавања. Упутства за инсталацију су доступна на веб-сајту алата. Тренутно не постоји туторијал. *Cppcheck* је постигао 8 од 16 бодова у овом сету карактеристика.
3. Што се тиче подршке за циклус прегледа кôда, алат *Cppcheck* обезбеђује преглед свих грешака у изворном кôду програма приликом генерисања извештаја. Праћење исправљених грешака у пројекту могуће је само поновним покретањем алата над изворним кôдом. Међутим, алат приказује где се грешка налази у изворном кôду и предлаже могућа решења за откривену грешку. *Cppcheck* индицира грешке са ознаком *CVE* (*Common Vulnerabilities and Exposures*), али не наводи процену нивоа озбиљности грешке нити време потребно за њено отклањање. Овај алат је постигао 8 од 13 бодова у овом сету карактеристика.
4. Управљање процесом: *Cppcheck* дозвољава рад више корисника, али овај режим не подржава праћење ажурирања. Алат омогућава рад на више пројеката истовремено и садржи неке корисне функције за управљање документима. *Cppcheck* нема подршку за управљање улогама. Алат имплементира систем пријављивања који захтева корисничко име и лозинку. Овај алат је постигао 10 од 16 бодова у овом сету карактеристика.

Коначно, бодови које је остварио алат *Cppcheck* приказани су у табели 10.

Табела 10 – Резултати анализе алата *Cppcheck*

Сет карактеристика	Подкарактеристике	Тежинска оцена	Оцена сета карактеристика	% задовољења критеријума
F1	*F1-SF01*	3	6/6	100%
	F1-SF02	3		
F2	*F2-SF01*	2	9.5/16	59.38%
	F2-SF02	3		
	F2-SF03	3		
	F2-SF04	0		
	F2-SF05	1.5		
F3	*F3-SF01*	2	8/13	61.54%
	F3-SF02	0		
	F3-SF03	3		
	F3-SF04	3		
	F3-SF05	0		
	F3-SF06	0		
F4	*F4-SF01*	2	10/16	62.5%
	F4-SF02	2		
	F4-SF03	2		
	F4-SF04	0		
	F4-SF05	4		
УКУПАН РЕЗУЛТАТ			**Укупан резултат у процентима**	
33.5/51			**65.69%**	

У наставку биће приказан преглед резултата бодовања за alat *FindBugs,* док је преглед резултата које је остварио овај алат приказан у табели 11.

1. Економичност: Алат *FindBugs* је бесплатан за коришћење и може се преузети са веб-сајта развојног тима. Међутим, алат је делимично одржаван, није ажуриран од марта 2020. године, и обезбеђује само једну контакт тачку за кориснике у случају да им је потребна помоћ. *FindBugs* је постигао 3 од 6 бодова у овом сету карактеристика.
2. Лакоћа инсталације: *FindBugs* се може преузети са веб-сајта развојног тима, инсталирати локално или додати у интегрисано развојно окружење (енгл. *integrated development environment – IDE*). Алат има делимично једноставан процес подешавања. Упутства за инсталацију су доступна на веб-сајту алата, али нема туторијала. *FindBugs* је постигао 8 од 16 бодова у овом сету карактеристика.
3. Када је у питању подршка за циклус прегледа кôда, алат *FindBugs* обезбеђује преглед свих грешака у изворном кôду програма приликом генерисања извештаја. Праћење исправљених грешака у пројекту није могуће у оквиру једног покретања алата. *FindBugs* показује где се грешка налази у изворном кôду (у извештају о грешкама, а не директно у кôду). *FindBugs* индицира грешке и ниво озбиљности грешке, али не процењује време потребно за њено отклањање. Овај алат је постигао 6 од 13 бодова у овом сету карактеристика.

4. *FindBugs* не подржава рад више корисника. Алат омогућава рад на више пројеката истовремено и може се додати у *IDE*, што олакшава праћење грешака у изворном кôду. *FindBugs* подржава управљање улогама. Овај алат имплементира систем пријављивања који захтева корисничко име и лозинку. *FindBugs* је постигао 10 од 16 бодова у овом сету карактеристика.

Табела 11 – Резултати анализе алата *FindBugs*

Сет карактеристика	Подкарактеристике	Тежинска оцена	Оцена сета карактеристика	% задовољења критеријума
F1	*F1-SF01*	3	3/6	50%
	F1-SF02	3		
F2	*F2-SF01*	2	8/16	50%
	F2-SF02	3		
	F2-SF03	3		
	F2-SF04	0		
	F2-SF05	1.5		
F3	*F3-SF01*	2	6/13	46.15%
	F3-SF02	0		
	F3-SF03	3		
	F3-SF04	3		
	F3-SF05	0		
	F3-SF06	0		
F4	*F4-SF01*	2	11/16	68.75%
	F4-SF02	2		
	F4-SF03	2		
	F4-SF04	0		
	F4-SF05	4		
УКУПАН РЕЗУЛТАТ			**Укупан резултат у процентима**	
28/51			**54.9%**	

У наставку биће приказан преглед резултата бодовања за alat *SonarQube,* док је преглед резултата које је остварио овај алат приказан у табели 12.

1. Економичност: све основне верзије алата *SonarQube* су бесплатне за коришћење и могу се преузети са веб-сајта развојног тима. Међутим, нису све функционалности овог алата доступне у основној верзији, већ само неке. Алат се добро одржава, редовно се ажурира (последње ажурирање било је у октобру 2024. године), и обезбеђује једну контакт тачку за кориснике у случају да им је потребна помоћ. *SonarQube* је постигао 4.5 од 6 бодова у овом сету карактеристика.
2. Лакоћа инсталације: *SonarQube* се може преузети са веб-сајта развојног тима и инсталирати локално. Алат има сложен процес подешавања. Да би се подесила пуна верзија *SonarQube*-а, потребно је инсталирати низ спољашњих технологија у зависности од програмског језика који се анализира. *SonarQube* је постигао 8 од 16 бодова у овом сету карактеристика.

3. Када је реч о подршци за циклус прегледа кôда, алат *SonarQube* обезбеђује преглед свих грешака у изворном кôду програма приликом генерисања извештаја. Праћење исправљених грешака у пројекту могуће је само поновним покретањем алата над изворним кôдом. Међутим, алат показује где се грешка налази у изворном кôду и предлаже могућа решења за откривену грешку. *SonarQube* за сваку грешку индицира одређени ниво озбиљности и величину грешке, као и процењено време потребно за исправљање грешке. Овај алат је постигао 10 од 13 бодова у овом сету карактеристика.
4. *SonarQube* подржава рад више корисника, али овај режим не подржава праћење ажурирања. Алат омогућава рад на више пројеката истовремено и садржи низ корисних функција за управљање документима. Алат се може додати у *IDE*, што олакшава праћење грешака у изворном кôду. *SonarQube* подржава управљање улогама. Овај алат имплементира сигурносни систем пријављивања који захтева корисничко име и лозинку при свакој посети. *SonarQube* је постигао 13 од 16 бодова у овом сету карактеристика.

Табела 12 – Резултати анализе алата *SonarQube*

Сет карактеристика	**Подкарактеристике**	**Тежинска оцена**	**Оцена сета карактеристика**	**% задовољења критеријума**
F1	*F1-SF01*	3	4.5/6	75%
	F1-SF02	3		
F2	*F2-SF01*	2	8/16	50%
	F2-SF02	3		
	F2-SF03	3		
	F2-SF04	0		
	F2-SF05	1.5		
F3	*F3-SF01*	2	10/13	76.92%
	F3-SF02	0		
	F3-SF03	3		
	F3-SF04	3		
	F3-SF05	0		
	F3-SF06	0		
F4	*F4-SF01*	2	13/16	81.25%
	F4-SF02	2		
	F4-SF03	2		
	F4-SF04	0		
	F4-SF05	4		
УКУПАН РЕЗУЛТАТ			**Укупан резултат у процентима**	
35.5/51			**69.61%**	

Табела 13 представља сажете резултате за сва три алата коришћена у овом истраживању.

Табела 13 – Преглед резултата свих алата по сетовима карактеристика

Алат	*F1*	*F2*	*F3*	*F4*	Укупно
CppCheck	100%	59.4%	61.5%	62.5%	**65.6%**
FindBugs	50%	50%	46.1%	68.7%	**54.9%**
SonarQube	75%	50%	76.9%	81.2%	**69.6%**

SonarQube је постигао најбоље резултате у овој анализи, са 69.61% укупног резултата. Разлике у процентима осталих алата нису претерано велике, што указује на то да су одабрани алати са релативно сличним могућностима и опцијама за софтвер инжењере. У поређењу са друга два алата који су евалуирани у овом истраживању, један од највећих недостатака *SonarQube*-а је у *F1*. *SonarQube* је бесплатан само у основној верзији, док су *Cppcheck* и *FindBugs* отвореног кôда у пуним верзијама.

Недостаци свих алата уочени су приликом посматрања *F2* – ниједан од представљених алата не садржи званичан туторијал који описује поступак подешавања и коришћења алата. Такође, ниједан алат не омогућава праћење исправки грешака без поновног покретања алата.

На основу представљених резултата алат *SonarQube* изабран је за коришћење у склопу ИМ1.

3.2 Методологија обраде података истраживачког модела 2 (ИМ2)

Истраживачки модел 2 садржи хипотезе које претпостављају постојање значајне везе између пробема плагијаризма и процеса оцењивања софтверских пројеката. Поред тога испитује се и потреба и значај за увођењем алата за аутоматско оцењивање, заснованих на статичкој анализи кôда, у процесе оцењивања софтверских пројеката. Метода прикупљања података за ИМ2 је упитник. Потребно је дефинисати дизајн упитника, који се дефинише кроз одређивање његових димензија, где се мерна димензија односи на дефинисање конструката упитника и њихових типова података, а репрезентациона димензија на одређивање циљне популације упитника [102]. Стога су у овом потпоглављу представљени развој и дистрибуција мерног иструмента и његових конструката, опис циљне популације, односно узорка, као и статистичке методе примењене ради тестирања хипотеза и социо-демографска структура испитаника

3.2.1. Развој и дистрибуција мерног иструмента

У поглављу 2.2.3 описан је систематски преглед стања у области плагијаризма и аутоматског оцењивања софтверских пројеката. Наведеним примарним студијама истакнут је проблем плагијаризма у поступку оцењивања студентских пројеката, као и све већа потреба за алатима за аутоматско оцењивање софтверских пројеката. Са циљем тестирања хипотеза (Х10 – Х13) у склопу ИМ2, креиран је упитник као мерни иструмент. Упитник је наведен као прилог Б.

За формирање понуђених одговара на упитник, коришћена је Ликертова петостепена скала. Ликертова скала је најчешће коришћен тип психометријске рангирајуће скале која има способност да мери ставове, мишљења или перцепције испитаника на одређену тему [103], [104]. Скала представља низ тврдњи или ставки димензија упитника, а испитаници означавају одређени интервални ниво који одговара њиховој перцепцији. На тај начин, Ликертова скала пружа структуирани начин квантификовања квалитативних података и прикупљања људских мишљења.

Прва секција, односно уводни део упитника објашњава испитаницима мотивацију и значај истраживања и садржи основне информације о структури упитника и времену потребном да се испуни. Испитаници су обавештени да је упитник анонимног карактера и да ће се подаци користити искључиво у сврхе истраживања представљене докторске дисертације.

Друга секција – демографија, садржи значајна демографска питања и питања која се односе на афилијацију испитаника. С обзиром на то да је упитник намењен искључиво наставницима у области софтверског инжењерства, прво питање подразумева навођење афилијације испитаника.

Друго питање подразумева проверу испитаника у контексту циљне групе којој је намењен овај упитник, односно да ли држи неки облик наставе (предавања, часове или вежбе) у области софтверског инжењерства.

Треће питање из ове секције односи се на ниво познавања области статичке анализе кôда и употребе алата за статичку анализу кôда. Ово питање нудило је понуђене одговоре из Ликтерове скале, где је:

1 – „слабо“, 2 – „умерено добро“, 3 – „добро“, 4 – „веома добро“ и 5 – „одлично“.

Друга секција односи се на употребу статичке анализе кôда и начине провере плагијаризма. Прво питање друге секције испитује да ли испитаници прегледају студентске софтверске пројекте применом статичке анализе кôда, понуђени одговори на ово питање су „да“ и „не“.

Друго питање из ове секције проверава у којој мери испитаници подстичу студенте да користе статичку анализу кôда док раде на пројектима. За одговор на ово питање коришћена је петостепена Ликтерова скала која подразумева одговоре:

1 – „не уопште“, 2 – „врло мало“, 3 – „понекад“, 4 – „често“, 5 – „врло често“.

Треће питање проверава које алате за статичку анализу кôда испитаници најчешће користе и подстичу своје студенте да користе. Понуђени су сви најпознатији алати за статичку анализу кôда, а остављен је и простор да испитаници сами наведу одговор уколико алат који користе није наведен на листи.

Четврто питање се односи на то да ли испитаници проверавају плагијаризам у студентским пројектима. Идеја овог питања јесте да се обухвате сви механизми за проверу плагијаризма у кôду. Понуђени одговори су „да“ и „не“.

Пето питање проверава да ли испитаници, уколико врше проверу плагијаризма, врше алатима и које алате најчешће користе. У овом питању су такође понуђени најпознатији алати за проверу плагијаризма, а остављен је и простор да испитаници сами наведу алат уколико се алат који користе не налази на листи.

Последње питање у овој секцији се односи на важност провере плагијаризма на пројектима у склопу курсева у области софтверског инжењерства. Питање гласи, „Колико сматрате важан проблем плагијаризма у студентским софтверским пројектима?“. На ово питање одговори су понуђени Ликтеровом скалом, на следећи начин:

1 – „није важно“, 2 – „мало је важно“, 3 – „делимично важно“, 4 – „важно“, 5 – „врло важно“.

Трећа секција, односи се на аутоматско оцењивање и генерални утисак о коришћењу алата у процесу оцењивања софтверских пројеката.

Прво питање треће секције проверава у којој мери испитаници сматрају да је изазовно прегледати велики број студентских софтверских пројеката без употребе алата за проверу плагијаризма и алата за аутоматско оцењивање. Понуђен одговори на ово питање су базирани на Ликтеровој скали, где је:

1 – „није изазовно“, 2 – „мало изазовно“, 3 – „делимично изазовно“, 4 – „изазовно“, 5 – „врло изазовно“.

Друго питање односи се на степен спремности испитаника да користе алат који би им омогућавао детекцију плагијаризма и аутоматско оцењивање софтверских пројеката. Такође, за одговоре коришћена је петостепена Ликтерова скала:

1 – „не бих користио“, 2 – „користио бих у мањој мери“, 3 – „делимично бих користио, 4 – „користио бих“, 5 – „користио бих у већој мери“.

Последње питање ове секције, а уједно и крај упитника односи се на то колико значајним и релевантним испитаници сматрају предложени метод за оцењивања софтверских пројеката. Под предложеним методом подразумева се употреба алата за оцењивање пројеката и проверу плагијаризма заснованог на статичкој анализи кôда. За одговор на ово питање коришћена је Ликтерова скала, где је:

1 – „незначајно и нерелевантно“, 2 – „мало значајно и релевантно“, 3 – „делимично значајно и релевантно“, 4 – „значајно и релеватно“, 5 – „врло значајно и релевантно“.

3.2.2. Демографске карактеристике испитаника

Репрезентациона димензија истраживачког дизајна ИМ2 се састоји од професора и асистената који учествују у наставном процесу у области софтверског инжењерства. Узорковање са сврхом (енгл. *purposive sampling*) је врста дефинисања узорка истраживања када постоји потреба да циљани учесници у истраживању поседују одређене квалитете, попут знања или искуства у некој области [103]. Узорковање са сврхом унапређује ригорозност истраживања, веродостојност прикупљених података и дубину разумевања истраживања од стране испитаника [105], [106]. Стога је приступ узорковања примењен у овом истраживању узорковања са сврхом, са укључивањем тоталне популације која испуњава критеријуме.

Упитник је креиран помоћу алата *SurveyMonkey*, који се користи за интернет упитнике и дистрибуиран је електронским путем [107], унапред дефинисаним редоследом. Позив на учешће у истраживању, заједно са линком ка електронском упитнику је послат свим потенцијалним учесницима 15. марта 2024. године. Подсетници су слати у три итерације, у размаку од недељу дана. Упитник је затворен 15. априла 2024. године. Попуњавање упитника и учешће у истраживању је било добровољно.

Од укупно 278 контактираних потенцијалних учесника у истраживању, 152 је приступило електронском упитнику, док је 124 попунило комплетан упитник. Стога, може се закључити да је стопа одговора била 44.6%. Како би се осигурао квалитет резултата истраживања насталих на основу обраде података прикупљених упитником, учесници који су означили да нису наставници у области софтверског инжењерства су искључени из узорка. На основу тога, коначан број одговора испитаника који је анализиран ради обраде података је 121.

Државе у којима је спровођено истраживање су Португал, Немачка, Хрватска, Северна Македонија, Босна и Херцеговина, Црна гора и Србија. У Србији је већи број Универзитета учествовао у истраживању, и то:

- Универзитет у Београду,
- Универзитет у Крагујевцу,
- Универзитет у Новом Саду.

Из преосталих држава, Универзитети који су учесвовали у истраживању су:

- Универзитет у Лубецку (Лубецк, Немачка),
- Универзитет НОВА ИМС (Лисабон, Португал),
- Универзитет у Загребу (Загреб, Хрватска),
- Универзитет у Љубљани (Љубљана, Словенија),
- Универзитет у Скопљу (Скопље, Северна Македонија),
- Универзитет у Бања Луци (Бања Лука, Босна и Херцеговина) и
- Универзитет у Подгорици (Подгорица, Црна Гора).

3.2.3. Обрада података

Фаза обраде података је кључна за припрему скупа података за даљу анализу и обезбеђивање да су сви подаци погодни за статистичку обраду. Обрада података обухватила је неколико корака:

1. чишћење података,
2. кодирање променљивих и
3. анализу расподеле података.

Кључне метрике, које се односе на истраживачко питање 3, описане су у наставку, а њихови скраћени називи (табела 14) биће коришћени у табелама и графичким приказима ради лакшег разумевања.

Табела 14 – Скраћени називи метрика из ИМ2

Метрика	Скраћени назив
Обим употребе алата за статичку анализу кôда код студената	ОУСАК
Важност проблема плагијата на предметима у области софтверског инжењерства	ПП
Обим употребе алата за проверу плагијаризма	ОУАПП
Изазовност прегледа великог броја пројеката без алата за проверу плагијата и квалитета кôда	ИПБА
Спремност наставника за коришћење алата за процену квалитета кôда и детекцију плагијата	СНКА
Адекватност метода процене који укључује алате за проверу плагијата и квалитета кôда	АМП

На почетку је извршено чишћење података како би се уклониле недостајуће и некоректне вредности. Подаци су прикупљени путем упитника, што је захтевало проверу свих записа ради конзистентности. Пројекти са недостајућим вредностима за кључне метрике (нпр. обим употребе алата за статичку анализу или процена адекватности метода) су елиминисани из анализе како би се обезбедила потпуна подршка статистичкој обради.

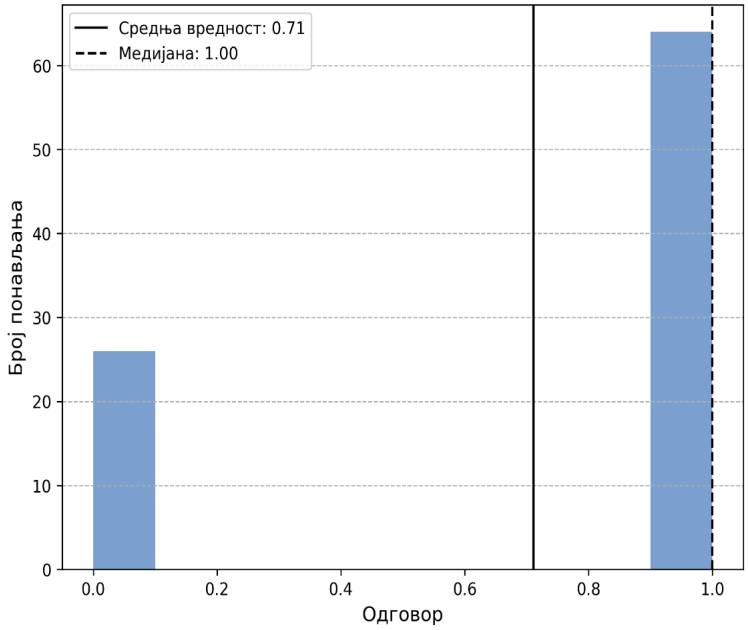
С обзиром на то да подаци нису нормално распоређени, коришћене су неке алтернативне метрике за описивање расподеле и централне тенденције података. У наставку су приказане основне статистичке мере (медијана, опсег и квартили) за сваку од испитиваних променљивих.

Након чишћења података, следећи корак је био кодирање бинарних променљивих у одговарајући формат како би биле погодне за статистичку анализу.

Бинарна променљива: Променљива која представља бинарну одлуку, коришћење алата за проверу плагијаризма кôда (ОУАПП), кодирана је на следећи начин:

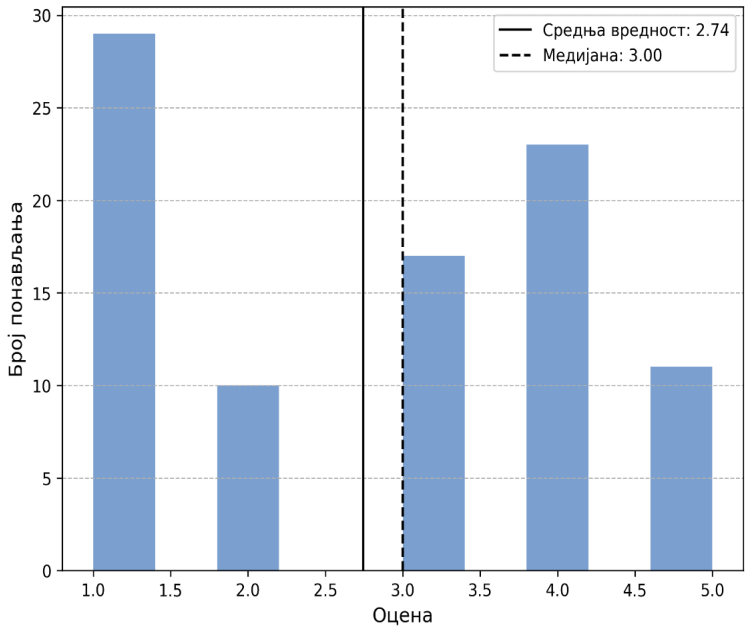
- 1 наставници користе алате за проверу плагијаризма у кôду.
- 0 наставници не користе алате за проверу плагијаризма у кôду.

Дескриптивна статистика за ову бинарну променљиву приказана је на слици 20.

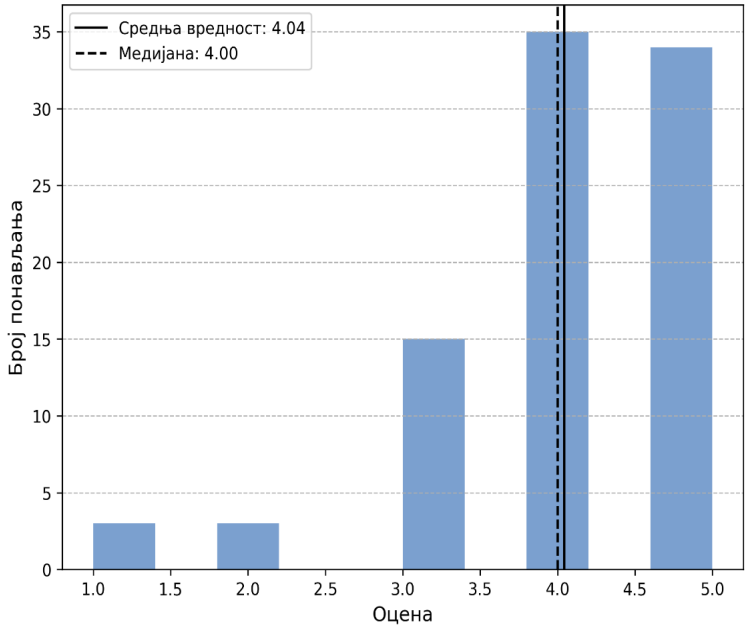


Слика 20 – Расподела броја појављивања метрике употребе алата за проверу плагијаризма у процесу оцењивања софтверских пројеката (ОУАПП)

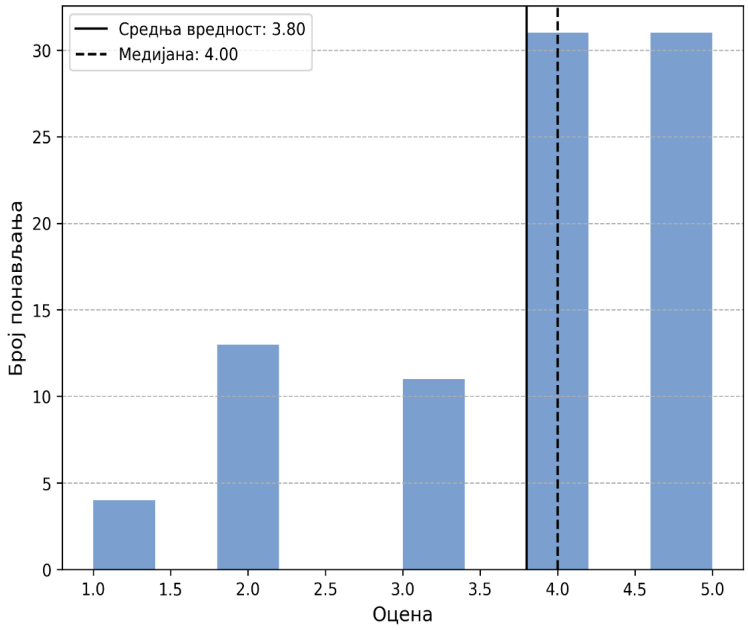
Остале метрике су ординалног типа (дефинисане према већ описаној Ликертовој скали) и њихове дескриптивне статистике ће бити приказане у наставку (Слика 21-25).



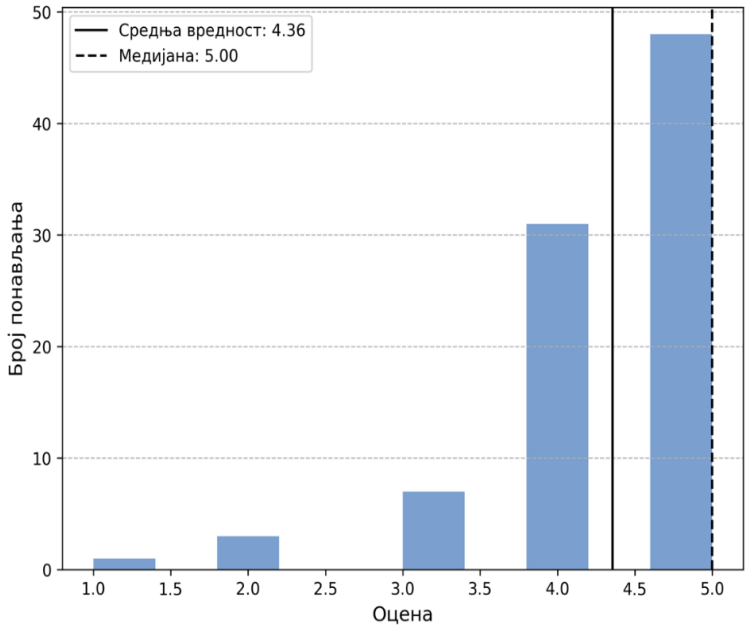
Слика 21 – Расподела броја појављивања метрике оцене у којој мери наставници подстичу студенте да користе алате за статичку анализу кôда у току развоја софтверских решења (ОУСАК)



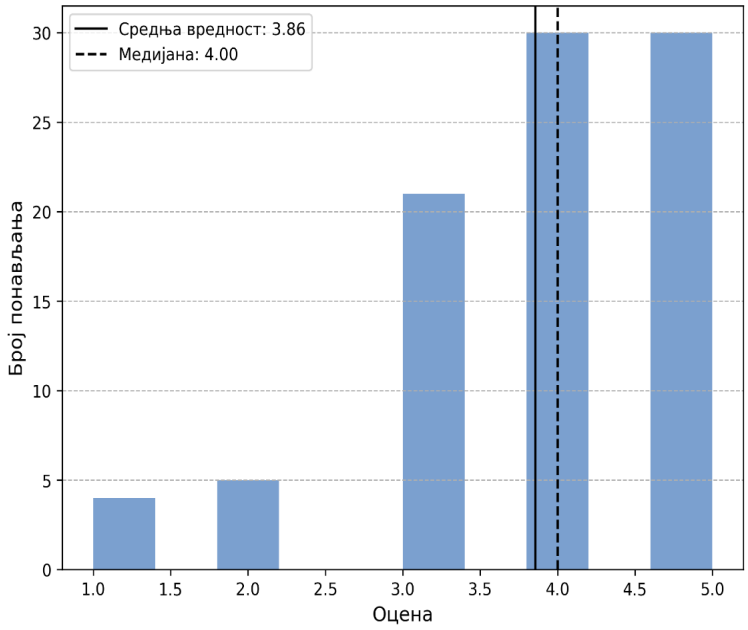
Слика 22 – Расподела броја појављивања метрике оцене важности проблема плагијарзима у области софтверског инжењерства (ПП)



Слика 23 – Расподела броја појављивања метрике оцене изазовности прегледа великог броја софтверских пројеката без употребе алата (ИПБА)



Слика 24 – Расподела броја појављивања метрике оцене спремности наставника за коришћење алата за процену квалитета кôда и детекцију плагијата (СНКА)



Слика 25 – Расподела броја појављивања метрике оцене валидности и адекватности алата за проверу плагијаризма и оцену квалитета кôда у процесу оцењивања (АМП)

3.2.4. Избор статистичких тестова и метода

Ово поглавље пружа преглед статистичких тестова одабраних за тестирање хипотеза постављених у оквиру ИМ2. Будући да су подаци у истраживању прикупљени из упитника и да испитиване варијабле не прате нормалну расподелу, изабрани су непараметарски тестови који не захтевају претпоставку о нормалности података. Наведени тестови омогућавају објективну анализу асоцијација и разлика међу групама и варијаблама у истраживању.

На основу специфичности сваке хипотезе, примењени су статистички тестови наведени у наставку.

Спирманов коефицијент корелације – примењен је за тестирање веза између рангираних варијабли, нарочито када су у питању ставови наставника у односу на изазове у прегледу пројеката и њихова спремност за коришћење алата.

Хипотеза X10, која испитује повезаност значаја проблема плагијата и спремности наставника за коришћење алата, као и X11, која се односи на везу између изазовности прегледа великог броја пројеката без алата и спремности за коришћење алата. Спирманов коефицијент корелације коришћен је за процену јачине и правца монотоних асоцијација између две рангиране променљиве. Пошто није потребна нормална расподела, овај тест је погодан за податке који показују одређену асиметрију или дискретну расподелу.

Mann-Whitney U тест је коришћен за поређење разлика у ставовима између две независне групе, нарочито између наставника који користе алате за проверу плагијата и оних који их не користе.

Mann-Whitney U тест се користи за испитивање разлике у медијанама две независне групе. Овај тест је погодан када су подаци редни или када не прате нормалну расподелу, што га чини одговарајућим за поређење група у овом истраживању.

Хи-квадрат (χ^2) тест примењен је за утврђивање асоцијације између категоријалних варијабли, посебно у случају ставова наставника у погледу адекватности алата за аутоматско оцењивање и њихове спремности за коришћење алата.

Хи-квадрат (χ^2) тест независности коришћен је за анализу асоцијације између две категоријалне варијабле. У овом контексту, тест омогућава процену повезаности између ставова наставника о коришћењу алата и њихове спремности за прихватање алата у оцењивању студентских пројеката.

Применом наведених статистичких тестова обезбеђена је детаљна и објективна анализа постављених хипотеза. Одабрани тестови одговарају врсти података и структури истраживања, омогућавајући да се одговори на истраживачка питања кроз мерење повезаности и разлика међу испитиваним променљивама.

4. Резултати истраживања

Резултати истраживања су представљени кроз резултате који се тичу истраживачког модела 1 и резултате који се тичу истраживачког модела 2.

4.1 Резултати истраживачког модела 1

Резултати истраживачког модела 1 представљени су кроз резултате који се односе на анализу утицаја представљених фактора поставке пројекта на квалитет кôда, резултате којим је испитивана корелација између квалитета кôда и академске успешности коју студенти постижу. Такође, описана је и валидација резултата употребом издвојеног сета података из друге институције.

4.1.1. Резултати анализе утицаја фактора поставке пројекта на квалитет кôда

У овом поглављу представљени су резултати анализе утицаја различитих фактора поставке пројеката на квалитет кôда (ИМ1). Користећи статистичке технике, попут биномне логистичке регресије, истражено је како поједини фактори, као што су избор технологије, примена алата за статичку анализу кôда, величина пројекта и други, утичу на метрике квалитета кôда. Циљ ове анализе је био да се идентификује који од фактора имају статистички значајан утицај на смањење или повећање квалитета кôда. У наставку су приказани резултати корелација за сваки од испитиваних фактора и њихов утицај на коначан квалитет пројеката. Резултати би требало да омогуће идентификацију фактора поставке пројекта који имају највећи потенцијал да побољшају технички квалитет кôда у софтверским пројектима. Кôд спроведених метода обраде података и статистичких анализа описан је у прилогу В.

Резултати анализе показали да постоји статистички значајна позитивна корелација између УСАК и ИТ са КК, док је ВП показала статистички значајну негативну корелацију са КК. Сви остали независни фактори нису показали статистички значајну корелацију са КК на нивоу значајности од 5%.

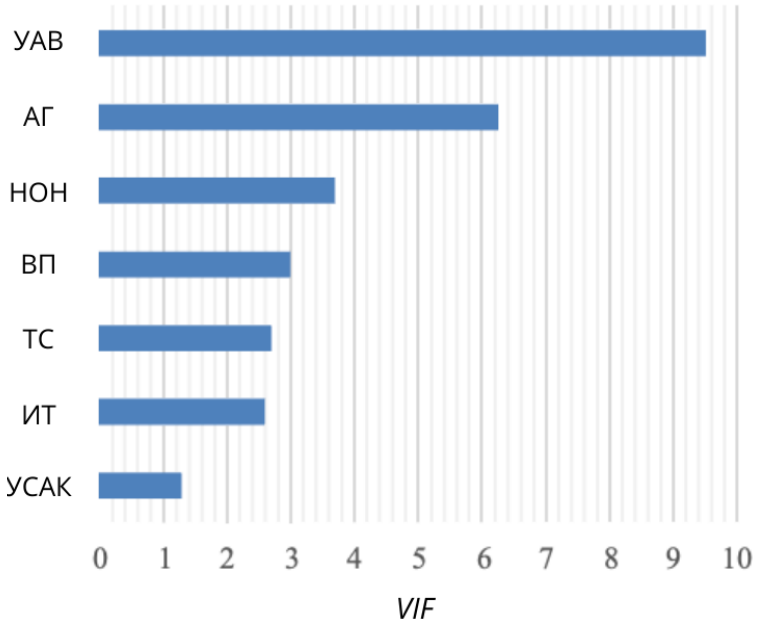
Табела 15 приказује резултате корелације између различитих фактора поставке пројекта и квалитета кôда (КК), користећи статистички параметар *Тау* за мерење јачине корелације и *p*-вредност за проверу њене статистичке значајности. У овом случају, позитивна корелација између УСАК (коришћење алата за статичку анализу) и ИТ (избор технологије) са КК показује да наведени фактори позитивно утичу на квалитет кôда, док негативна корелација са ВП (величина пројекта) указује на то да већи пројекти доводе до смањења квалитета кôда. Вредност *p* указује на статистичку значајност корелације, где су вредности мање од 0.05 сматране значајним. Ово значи да су корелације УСАК, ИТ и ВП са КК статистички значајне, док остали фактори (УАВ, АГ, ТС, НОН) нису показали значајне резултате.

Табела 15 – Корелација између поставке пројекта и квалитета кôда

Варијабле	*Tau*	*p*
УСАК	0.0543	0.0222
УАВ	0.0748	0.0925
ВП	-0.0954	0.0417
АГ	0.0380	0.3019
НОН	0.0632	0.1551
ТС	-0.0405	0.3625
ИТ	0.1281	0.0105

VIF или *Variance Inflation Factor*, је показатељ који се користи у регресионим анализама за процену степена мултиколинеарности међу независним варијаблама у моделу. Мултиколинеарност настаје када су две или више независних варијабли високо корелисане међусобно, што може довести до проблема у интерпретацији резултата регресије. Висок степен мултиколинеарности отежава тачну процену индивидуалног утицаја сваке променљиве на зависну променљиву, јер су ефекти различитих варијабли тешко раздвојиви.

Вредност *VIF*-а показује колико је варијанса коефицијента одређене независне променљиве превисоко процењена услед корелације са другим независним променљивама у моделу. Уобичајено правило је да *VIF* вредности испод 5 указују на прихватљив ниво мултиколинеарности, док вредности изнад 5 указују на потенцијалне проблеме. У овом случају, све варијабле које су показале статистички значајну корелацију са КК (УСАК, ИП, ВП) имале су *VIF* испод 5, што значи да мултиколинеарност није проблем за ове променљиве и да је процена њихових ефеката на КК поуздана. С друге стране, УАВ (употреба алата за контролу верзија) и АГ (академска година) су имале *VIF* изнад 5, што сугерише да постоји виша мултиколинеарност међу овим варијаблама, што може утицати на прецизност прорачунатих коефицијената за њих. Слика 26 приказује вредности *VIF* за све варијабле.



Слика 26 – *Variance Inflaction Factor (VIF)* за независне варијабле поставке пројекта

Анализа биномне логистичке регресије открила је статистички значајне узрочно-последичне везе између три кључне независне променљиве: коришћење алата за статичку анализу кôда (УСАК), величина пројекта (ВП), и избор технологије (ИТ) са квалитетом кôда (КК). Овај модел омогућава процену како свака од променљивих утиче на вероватноћу развоја висококвалитетног кôда.

Коришћење алата за статичку анализу кôда (УСАК) показало је позитивну и статистички значајну везу са КК. Резултати показују да употреба алата као што је *SonarQube* повећава шансе за развој висококвалитетног кôда за 1,31 пут (*OR* = 1.31). Ово значи да студенти који су користили алате за статичку анализу кôда имају око 31% већу вероватноћу да произведу кôд вишег квалитета у односу на оне који нису користили такве алате. Статистички значај ($p = 0.0120$) потврђује да је овај резултат валидан и да коришћење алата директно доприноси побољшању квалитета кôда.

Избор технологије (ИТ) такође је имао позитиван и статистички значајан утицај на квалитет кôда. Резултати показују да коришћење *.NET* и *Angular* или само *.NET* технологија повећава вероватноћу развоја висококвалитетног кôда за 1,68 пута (*OR* = 1.6844). Овај резултат сугерише да ове технологије пружају боље алате и праксе за развој софтвера, што омогућава студентима да развију кôд који је структурисанији и са мање грешака. Статистички значај ($p = 0.0003$) указује да је овај ефекат поуздан.

Величина пројекта (ВП) показала је негативну везу са КК, што значи да већи пројекти имају тенденцију за нижим квалитетом кôда. За сваки пораст стандардне девијације у броју линија кôда, вероватноћа развоја висококвалитетног кôда опада за око 44% (*OR* = 0.5604). Овакав резултат је последица повећане сложености и већег броја грешака које се јављају у већим пројектима. Такође, овакав резултат је статистички значајан ($p = 0.0000$), што указује да већи обим кôда доводи до већих изазова у одржавању квалитета.

Остале независне променљиве, као што су тип сарадње (ТС) и начин одржавања наставе (НОН), нису показале статистички значајну везу са квалитетом кôда у овом моделу, што се види у релативно високим *p*-вредностима (изнад 0.05). То значи да наведени фактори не утичу значајно на КК у оквиру овог истраживања.

Резултати биномне логистичке регресије представљени су у табели 16.

Табела 16 – Резултати биномне логистичке регресије

Варијабле	**Коефицијент**	**стд. грешка**	***z* вредност**	***Pr(>|z|)***	***OR***
УСАК	0.2679	0.1066	2.5117	0.0120	1.3072
ВП	-0.5790	0.1308	4.4272	0.0000	0.5604
НОН	0.1022	0.1138	0.8981	0.3691	1.1076
ТС	-0.1087	0.1491	-0.7288	0.4661	0.8970
ИП	0.5214	0.1455	-3.6453	0.0003	1.6844

Табела 16 пружа преглед резултата биномне логистичке регресије, укључујући коефицијенте и однос шанси (*OR*) за сваки фактор. Коефицијент показује смер и јачину утицаја сваке независне променљиве на КК, док *OR* показује колико се шансе за развој висококвалитетног кôда мењају са променом јединице сваке променљиве. Негативне вредности коефицијената (нпр. ВП) указују на смањење шанси за висок квалитет кôда, док позитивне вредности (нпр. УСАК, ИТ) указују на повећање шанси.

У циљу прецизније процене важности сваке независне променљиве у моделу биномне логистичке регресије, коришћен је метод пермутационе важности (енгл. *permutation importance*). Овај метод процењује колико свака променљива доприноси тачности предвиђања модела, тако што насумично мења вредности једне променљиве, док остале остају непромењене. Уколико ова пермутација значајно смањи прецизност модела, то сугерише да та променљива игра кључну улогу у тачности предвиђања.

Током 10 итерација ове анализе, установљено је да су три променљиве – УСАК (коришћење алата за статичку анализу кôда), ВП (величина пројекта) и ИТ (избор технологије) – имале сличан ниво значаја у предвиђању квалитета кôда (КК). То значи да свака од променљивих значајно доприноси моделирању резултата и да их није могуће занемарити без негативног утицаја на тачност предвиђања.

Табела 17 приказује вредности коефицијента *Tau*, које мере јачину корелације између независне променљиве и зависне променљиве (КК), као и *p*-вредности које показују да ли је та корелација статистички значајна. Све три променљиве (УСАК, ВП, ИП) имају *p*-вредност мању од 0.05, што указује на то да су корелације статистички значајне. Висок коефицијент *Tau* за ове три променљиве потврђује њихову јачу везу са КК у односу на остале факторе.

Табела 17 – Корелација између независних варијабли и КК

Варијабле	***Tau***	***p***
УСАК	0.1259	0.0000
ВП	0.1375	0.0039
ИТ	0.1259	0.0000

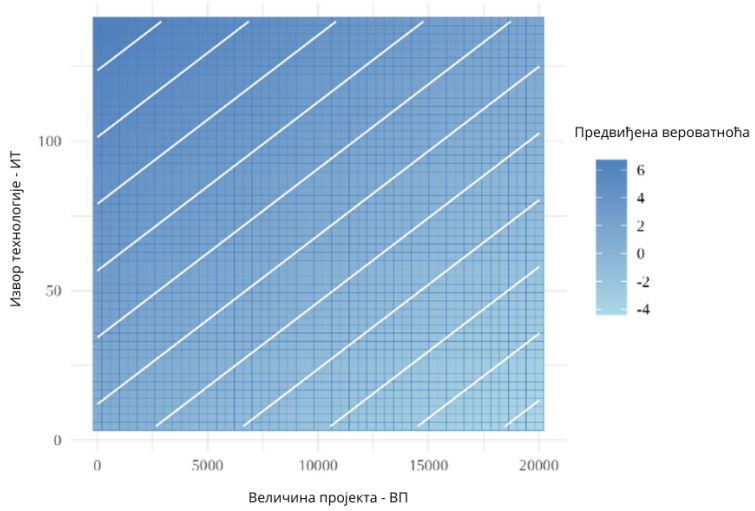
У овом случају, УСАК, ВП, и ИТ показују вредности *Tau* око 0.12, што указује на умерену позитивну корелацију. Ова корелација је статистички значајна, како је потврђено ниским *p*-вредностима (све су испод 0.05), што значи да је мала вероватноћа да су приказани резултати случајни.

Пермутациона анализа помаже да се боље разумеју фактори који највише утичу на модел и омогућава истраживачима да идентификују које променљиве имају највећи утицај на квалитет кôда. У овом случају, УСАК, ВП и ИТ се издвајају као најважнији фактори који доприносе предвиђању квалитета кôда, чиме овај модел добија на прецизности и поузданости.

У завршној фази овог дела анализе, истражен је синергетски ефекат три кључне независне променљиве – коришћење алата за статичку анализу (УСАК), величина пројекта (ВП) и избор технологије (ИТ) – на квалитет кôда (КК). Циљ је био да се утврди да ли комбинација ове три променљиве има већи утицај на квалитет кôда него што то има свака појединачно. Да би се ово постигло, примењен је нови модел биномне логистичке регресије који користи нетрансформисане податке како би се обезбедила боља интерпретабилност резултата. Парцијални графикони зависности су коришћени за визуализацију интеракција између променљивих и њиховог заједничког утицаја на КК.

Међу анализираним интеракцијама, јасан синергетски ефекат је утврђен између величине пројекта (ВП) и избора технологије (ИТ). Овај ефекат је посебно изражен када су се користиле технологије као што су *.NET* и *Angular* (или само *.NET*). Резултати су показали да је комбиновањем мањих пројеката (са мањим бројем линија кôда) и одабира наведених технологија значајно повећана вероватноћа развоја висококвалитетног кôда.

Слика 27 приказује овај синергетски ефекат, показујући како интеракција између величине пројекта и избора технологије утиче на вероватноћу развоја висококвалитетног кôда. На графикону је видљиво да мањи пројекти, када су реализовани помоћу модерних и добро подржаних технологија као што су *.NET* и *Angular*, значајно повећавају шансе за квалитетан кôд. Ова комбинација смањује сложеност пројекта и олакшава студентима да следе најбоље праксе развоја софтвера, што резултира мањим бројем неправилности, бољом структуром кôда и мањим бројем дизајнерских недостатака. Са друге стране, када је величина пројекта већа (више линија кôда), утицај технологије на КК је смањен, што указује на то да комплексност већих пројеката захтева додатне алате и ресурсе за одржавање квалитета.



Слика 27 – Синергетски ефекат величине пројекта и избора технологије на квалитет кôда

Овај резултат наглашава важност оптимизације величине пројеката и избора одговарајућих технологија за постизање најбољих могућих исхода у погледу квалитета кôда.

4.1.2. Анализа корелације између квалитета кôда и академске успешности

Применом *Mann-Whitney U* теста испитиване су разлике у оценама (ОС) између студената који су премашили праг квалитета кôда (КК) и оних који нису. Резултати су показали да студенти са вишим КК постижу нешто више медијалне оцене на испиту и имају мању варијансу у оценама у поређењу са студентима са нижим КК. Ова статистички значајна разлика потврђује да постоји узрочно-последична веза између високог квалитета кôда и бољих резултата које студенти постижу. *Cliff*-ова Делта, која је износила 0.3189, указује на умерену величину ефекта, што значи да студенти са бољим квалитетом кôда у просеку остварују боље резултате на испитима.

С друге стране, резултати *Mann-Whitney U* теста који су испитивали везу броја рокова потребних за полагање пројекта (БРП) и квалитета кôда (КК) нису показали статистички значајне разлике. Овај резултат сугерише да време које је студентима било потребно за завршетак пројекта није имало значајан утицај на квалитет кôда који су произвели.

Табела 18 приказује резултате *Mann-Whitney U* теста за две корелације: однос између квалитета кôда (КК) и оцена на испиту (ОС), и однос између КК и времена потребног за завршетак пројекта издраженог бројем рокова потребних за полагање пројекта (БРП).

Табела 18 – Резултати корелације између квалитета кôда и академског успеха студената

Варијабле	*W*	*p*
КК → ОС	26830	0.0029
КК → БРП	29562	0.1971

W у табели 18 је вредност *Mann-Whitney U* статистике, која представља резултат рангирања свих вредности у анализи. Већа вредност *W* значи да су рангирања једне групе у просеку већа од рангирања друге групе.

Ови резултати указују на то да постоји значајна веза између квалитета кôда и успеха на испиту, док време завршетка пројекта нема значајан утицај на сам квалитет кôда.

4.1.3. Валидација резултата

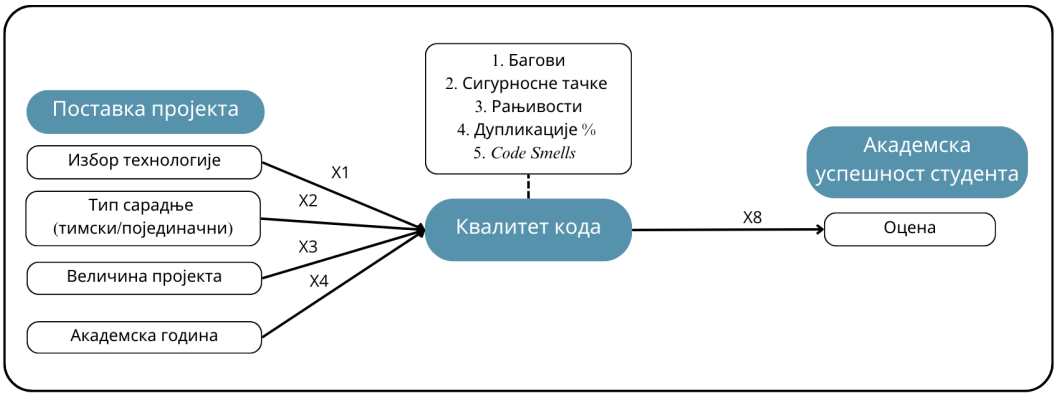
Резултати иницијалног истраживања добијени су на основу софтверских пројеката насталих на различитим студијским програмима у оквиру Факултета техничких наука Универзитета у Новом Саду.

Да би се валидирали добијени резултати, прикупљен је додатни скуп од 210 софтверских пројеката са Факултета електротехнике и рачунарства Универзитета у Загребу. У наставку ће бити описани подаци прикупљени из ове групе пројеката, као и резултати спроведених анализа.

Подаци су прикупљени у сарадњи са Универзитетом у Загребу, а добијени подаци о пројектима и сами пројекти се у одређеној мери разликују од иницијалног скупа пројеката.

Ограничења овог дела истраживања произилазе из уредби Универзитета у Загребу о правилима дељења података о студентима и студентским пројектима са другим институцијама. Поред наведених уредби, додатне разлике у односу на иницијални модел огледају се у различитом начину организовања пројеката на овој институцији.

У складу са наведеним, иницијални истраживачки модел 1 је измењен и приказан на слици 28.



Слика 28 – Измењен истраживачки модел 1 са циљем анализе података за валидацију

Као што је приказано на слици, у оквиру овог дела анализе, истраживачки модел је измењен и искључени су следећи чиниоци:

- Употреба алата за верзионисање (УАВ – хипотеза X5): У оквиру курсева није бележена информација о употреби алата за верзионисање.
- Употреба алата за статичку анализу кôда (УСАК – хипотеза X6): У оквиру курсева није бележена информација о употреби алата за статичку анализу кôда.

- Начин одржавања наставе (НОН – хипотеза X7): На свим пројектима коришћеним у истраживању, настава је одржавана уживо, па ова хипотеза није могла бити испитана.
- Број рокова потребних за полагање пројекта (БРП – хипотеза X9): Пројекти су организовани тако да постоји само један термин одбране, те ова хипотеза није могла бити испитана.

Подаци о осталим чиниоцима истраживања су прикупљени, обрађени и припадајуће хипотезе су на исти начин тестиране.

Као и у инцијалном истраживању, подаци су пре анализе обрађени и прочишћени на следећи начин:

- Недостајуће вредности: Пројекти са недостајућим вредностима за кључне метрике (нпр. багови, дупликације, рањивости), или недостајућим оценама су замењене вредношћу медијане, како би се обезбедила комплетност података.
- Дуплиране вредности: У случају да су постојале дуплиране вредности за исте пројекте, задржане су само оне вредности које су биле најкомплетније и најновије, како би се избегла погрешна анализа.

Након чишћења података, следећи корак је био кодирање номиналних и бинарних променљивих у одговарајући формат, како би биле погодне за статистичку анализу:

Бинарне променљиве: Променљиве које представљају бинарне одлуке, као што је тип сарадње (ТС – тимски или индивидуални рад), кодиране су на следећи начин:

- 1 за тимски рад и
- 0 за индивидуалан рад.

Номиналне променљиве: Номиналне променљиве које представљају избор технологије (ИТ) су кодиране на основу учесталости појављивања категорија у бази података. То значи да су технологије попут *wpf, Vue* и *Angular* добиле одређене нумеричке вредности у складу са учесталошћу њихове употребе у пројектима.

Оцене на предмету (ОС) скалиране су на следећи начин:

- „НП”, која означава студенте који нису положили предмет – скалирана на 0,
- оцене 6 и 7 – скалиране на 1,
- 8 – скалирана на 2,
- 9 и 10 – скалиране на 3.

Континуалне променљиве, које су представљале метрике квалитета кôда (нпр. број багова, проценат дуплирања, број *code smells*, рањивости и безбедносни пропусти) биле су на различитим скалама, што је могло утицати на резултате анализе. Због тога је извршена стандардизација променљивих, што је укључивало следеће кораке.

- Стандардизација се састојала од претварања вредности сваке променљиве у скалу са средњом вредношћу од 0 и варијансом од 1. Ово је омогућило да се сви подаци обрађују на истој скали, чиме се избегава да једна променљива доминира резултатима само због веће величине својих вредности.
- Дистрибуција података: У овом кораку је такође проверена дистрибуција свих континуалних променљивих како би се уочила било каква права или закошеност. За променљиве које су показале знатно одступање од нормалне дистрибуције, извршена је трансформација података како би се обезбедила симетричнија дистрибуција.

Коначно, варијабле поставке пројекта је неопходно прилагодити на сличну скалу како би се избегла пристрасност у анализи. У супротном, промењљиве са већим распоном могу имати несразмерно велики утицај на резултате. Примењена је техника робустног скалирања, која подразумева трансформацију сваке промењљиве тако да се центрира око њене медијане и скалира према интерквартилном распону (разлика између 75. и 25. перцентила). За сваку промењљиву X_i трансформисана вредност X'_i израчунава се по формули:

$$X'_i = \frac{X_i - X_{median}}{X_{IQR}}$$

Где је X_{IQR} интерквартилни распон. Робустно скалирање је отпорно на екстремне вредности јер користи медијану и интерквартилни распон уместо средње вредности и стандардне девијације, које су осетљиве на екстремне вредности.

Пример прикупљених и необрађених података у склопу истраживања које има за циљ валидацију резултата приказани су у оквиру прилога Г. У табели 19 приказана је дескриптивна статистика података коришћених за валидацију резултата.

Табела 19 – Дескриптивна статистика података из истраживања за валидацију резултата

Варијабла	Бр. података	Ср. вредност	*std*	мин.	25%	50%	75%	макс.
ОС	210.0	2.0	1.0	0.0	1.0	2.0	3.0	3.0
Б	210.0	120.3	216.8	0.0	0.0	1.0	115.8	681.0
Д%	210.0	4.7	6.8	0.0	0.0	0.0	11.2	24.4
Р	210.0	7.3	27.6	0.0	0.0	0.0	0.0	170.0
CS	210.0	655.1	1181.0	0.0	2.0	7.0	855.5	4200.0
БП	210.0	10.0	14.7	0.0	0.0	2.0	19.0	45.0
АГ	210.0	2.9	0.9	2.0	2.0	3.0	4.0	4.0
ВП	210.0	30440.7	53664.8	75.0	387.0	625.0	51875.0	163000.0
ТС	210.0	0.3	0.5	0.0	0.0	0.0	1.0	1.0
ИТ	210.0	73.3	15.4	53.0	56.5	67.0	90.0	90.0

На исти начин као и у инцијалном истраживању у склопу ИМ1, над овим метрикама креирана је нова промењљива квалитет кôда (КК) – креирање нове промењљиве описано је у поглављу 3.1.3.

Приликом спровођења статистичких тестова, коришћени су исти тестови као и у инцијалном истраживању описаном у поглављу 3.1.4, те кôд спровођења тестова над подацима није наведен као прилог у дисертацији.

Резултати тестирања хипотеза (Х1 – Х4) које испитују утицај фактора поставке пројекта на квалитет кôда софтверског пројекта, представљени су у табели 20.

Табела 20 – Резултати биномне логистичке регресије за хипотезе 1 – 4

Варијабла	Коефицијент	Стандардна грешка	*z*	*p*	Доња граница (95%)	Горња граница (95%)
ИТ	-9.5933	1.913	-5.014	0.000	-13.343	-5.843
ТС	-3.6151	1.372	-2.635	0.008	-6.304	-0.926
ВП	0.2387	0.537	0.445	0.656	-0.813	1.290
АГ	-6.0085	1.917	-3.135	0.002	-9.765	-2.252

На основу приказаних резултата испитивања корелације између различитих фактора поставке пројекта и варијабле КК, може се видети да је потврђен утицај варијабле избора технологије за развој пројекта (ИТ) на резултате квалитета кôда софтверског пројекта (КК), као и у инцијалном истраживању. На овај начин и у оквиру валидационог истраживања потврђена је хипотеза Х1.

Поред ове варијабле, статистички значајан резултат постигнут је и при анализи корелације између типа сарадње на пројекту (тимски/индивидуално – ТС) и квалитета кôда. На овај начин потврђена је хипотеза Х2 да тип сарадње у поставци пројекта има утицај на квалитет кôда софтверског пројекта. Овај резултат разликује се у односу на иницијално истраживање и биће даље дискутован у поглављу 5.

Остали фактори поставке пројекта нису показали статистички значајне утицаје на квалитет кôда.

Други део валидационог истраживања укључује проверу корелације између квалитета кôда и оцене (ОС) коју студент постиже на предмету. У табели 21 представљени су резултати испитивања овог утицаја дефинисаног у оквиру хипотезе Х8.

Табела 21 – Испитивање корелације између квалитета кôда и оцене студента

Варијабле	*W*	*p*
КК → ОС	2730.0	0.0000

Резултат добијен *Mann-Whitney U* тестом показује да је квалитет кôда у статстички значајној корелацији са оценом коју студент постиже на предмету. На тај начин и у оквиру валидационог истраживања потврђена је хипотеза Х8, као што је то био случај и у иницијалном истраживању.

4.1.4. Процена и предвиђање квалитета кôда пројеката на општем узроку

У претходном поглављу показано је да постоји јасна корелација између квалитета кôда и оцене коју студент добија на предмету. Односно, од 210 студената, 42 студента развило је висококвалитетан кôд и добило високу оцену. Истраживањем у оквиру овог поглавља се проверава да ли ова хипотеза важи и на општем узорку.

Коришћена је Бернулијева расподела, са функцијом расподеле вероватноћа случајне променљиве *X*, дефинисана на следећи начин:

$$f(x) = P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}$$

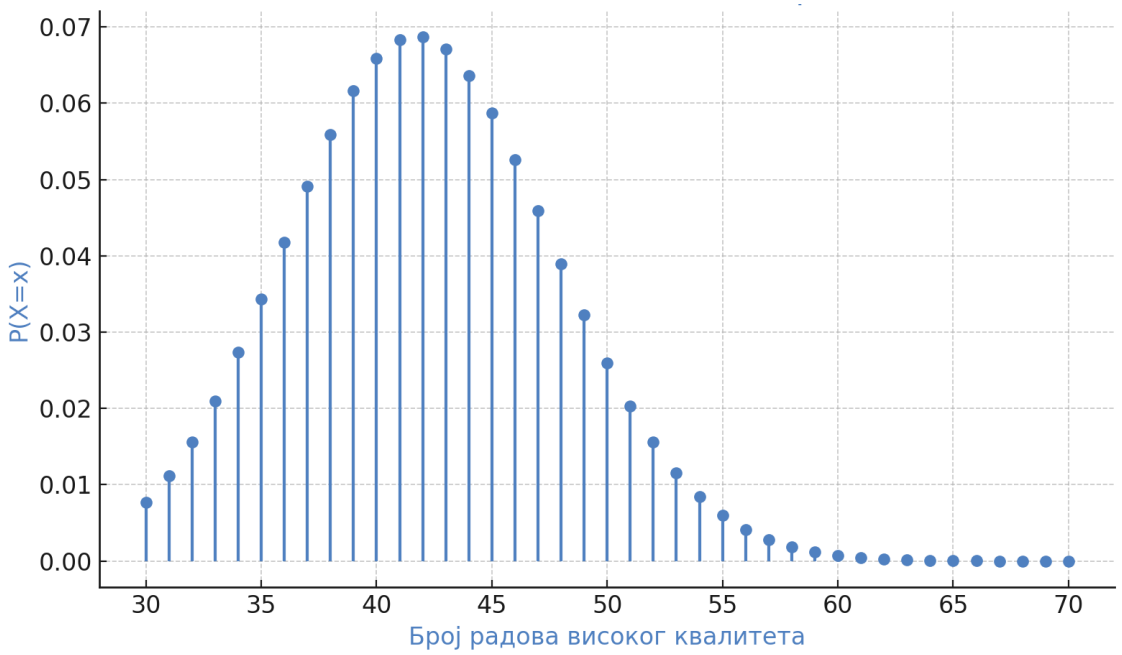
где је n број независних експеримената, p вероватноћа успеха, а x број успешних исхода.

Постављена су два главна параметра:

- $n = 210$, што представља укупан број студената у узорку,
- $p = 0.2$, вероватноћа да студент произведе кôд високог квалитета (добијена резултатима спроведеног истраживања)

Испитана је вероватноћа да ће се тачно 42 рада високог квалитета наћи у узорку исте величине и добијен је резултат од 6.87%.

На основу овога, испитана је расподела броја пројеката високог квалитета за опсеге од 30 до 70 пројеката и генерисан је графикон који показује расподелу вероватноћа (слика 29). Највећа вероватноћа је добијена за око 42 рада високог квалитета, што је у складу са очекиваном вредношћу Бернулијеве расподеле.

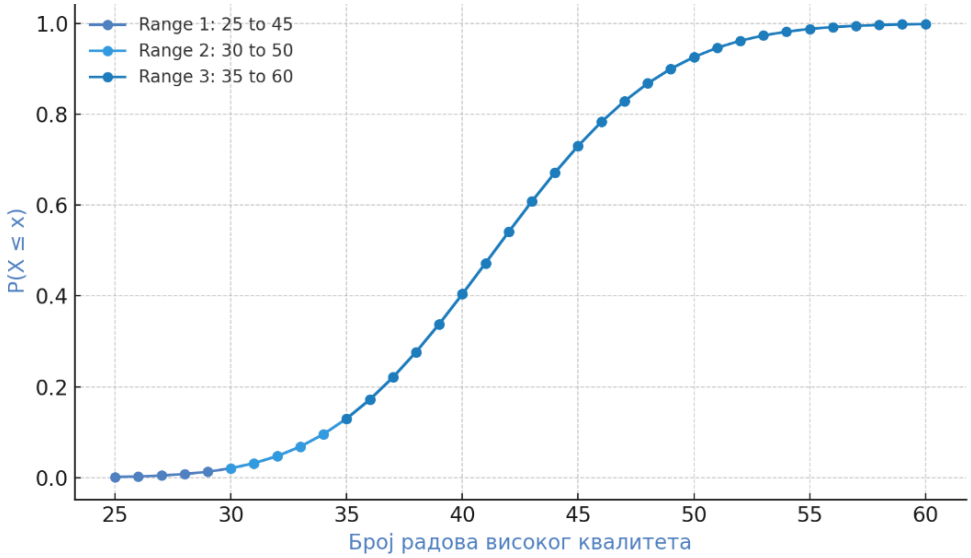


Слика 29 – Функција расподеле вероватноћа

Затим, испитана је и кумулативна дистрибутивна функција (енгл. *Cumulative distribution function – CDF*) за три опсега броја радова високог квалитета:

1. Опсег 25 до 45,
2. Опсег 30 до 50,
3. Опсег 35 до 60.

Кумулативна дистрибутивна функција, приказана на слици 30, показује вероватноћу да ће број радова високог квалитета бити мањи или једнак од одређене вредности. Графици су приказали постепену акумулацију вероватноће, што потврђује да је дистрибуција детерминистичка, омогућавајући интерполацију на било који узорак сличне величине.



Слика 30 – Кумулативна дистрибутивна функција

На основу спроведене анализе, утврђено је да постоји стабилна вероватноћа за појаву радова високог квалитета и да се добијени налази могу применити и на шире узорке. Ова анализа подржава хипотезу да квалитет кôда има значајан утицај на оцену коју студент добија. То је посебно важно у академским окружењима где се често поставља питање објективности и правичности оцењивања студентских пројеката. Добијени резултати сугеришу да је могуће применити сличне методе анализе за предвиђање квалитета кôда и у другим контекстима, што може олакшати процес оцењивања и допринети побољшању наставног процеса.

Коначно, ово истраживање пружа чврсту основу за развој алата за аутоматизовану процену кôда, који би могли бити интегрисани у образовне платформе ради олакшавања процеса оцењивања, о чему ће бити детаљније дискутовано у наредним поглављима.

4.2 Резултати истраживачког модела 2

Резултати истраживачког модела 2 представљени су кроз резултате који се односе на тестирање хипотеза помоћу тестова описаних у оквиру поглавља 3.2.3. У оквиру овог поглавља биће представљени резултати тестирања сваке хипотезе у склопу истраживачког модела ИМ2.

Х10: Наставници који сматрају проблем плагијата значајним показују већу спремност за коришћење алата за проверу плагијата и процену квалитета кôда.

За хипотезу Х10, анализирана је корелација између значаја проблема плагијаризма (ПП) и спремности наставника за коришћење алата (СНКА). Спирманова корелација омогућава процену степена и правца монотоне везе између варијабли, а добијени резултати показују позитивну и статистички значајну повезаност. Резултати спроведеног теста приказани су у оквиру табеле 22.

Табела 22 – Резултати корелације између ПП и СНКА

Варијабле	Коефицијент корелације (ρ)	p	Закључак
ПП и СНКА	0.333	0.0013	Постоји позитивна и статистички значајна корелација

Добијени Спирманов коефицијент корелације износи 0.333, што указује на позитивну везу средње јачине између значаја проблема плагијата и спремности за коришћење алата. Вредност корелације је у позитивном смеру, што значи да са порастом значаја који наставници придају проблему плагијата, расте и њихова спремност да користе алате за проверу плагијата.

Добијена *p*-вредност износи 0.0013, што је мање од нивоа значајности од 0.05. Ово указује да је корелација статистички значајна, односно да је мало вероватно да је добијена веза настала случајно. Закључујемо да постоји стварна, значајна веза између ове две променљиве.

Овај резултат сугерише да наставници који сматрају проблем плагијата значајним имају већу спремност за коришћење алата за проверу квалитета кôда и откривање плагијата. Ово може указивати на свест о томе да алати могу ефикасно помоћи у решавању проблема плагијата, посебно међу онима који тај проблем доживљавају као критичан у образовном процесу.

X11: Наставници који сматрају преглед великог броја студентских пројеката без алата изазовним показују већу спремност за коришћење алата за проверу плагијата и процену квалитета кôда.

За хипотезу X11, детаљније је анализирана корелација између изазовности прегледа великог броја пројеката без алата (ИПБА) и спремности наставника за коришћење алата (СНКА). Спирманова корелација омогућава процену степена и правца монотоне везе између варијабли, а добијени резултати показују позитивну и статистички значајну повезаност. Резултати анализе представљени су у табели 23.

Табела 23 – Резултати корелације између ИПБА и СНКА

Варијабле	**Коефицијент корелације (ρ)**	***p***	**Закључак**
ИПБА и СНКА	0.344	0.0009	Постоји позитивна и статистички значајна корелација

Добијени Спирманов коефицијент корелације износи 0.344, што указује на позитивну везу средње јачине између изазовности прегледа великог броја пројеката без алата и спремности за коришћење алата. Ова вредност указује на то да наставници који преглед пројеката без алата доживљавају као изазован показују већу спремност за коришћење алата.

Добијена *p*-вредност износи 0.0009, што је мање од нивоа значајности од 0.05. Ово значи да је корелација статистички значајна и да је мало вероватно да је добијена веза настала случајно. Закључујемо да постоји стварна, значајна повезаност између ове две променљиве.

Овај резултат указује на то да наставници који сматрају да је преглед великог броја пројеката без алата изазован имају већу спремност да користе алате за проверу квалитета кôда и откривање плагијата. Ово сугерише да се алати сматрају корисним средствима за лакше и ефикасније прегледање и процену студентских пројеката, посебно међу наставницима који се суочавају са већим оптерећењем у прегледу.

X12: Наставници који користе алате за проверу плагијата сматрају да је проблем плагијата значајнији у односу на наставнике који не користе те алате.

Хипотеза X12 тестирана је применом *Mann-Whitney U* теста, који је примењен за испитивање разлике у значају проблема плагијата (ПП) између наставника који користе алате за проверу плагијата и оних који их не користе. *Mann-Whitney U* тест омогућава

поређење рангираних вредности две независне групе, а добијени резултати указују на статистички значајну разлику у перцепцији значаја проблема плагијата између наведених група.

Табела 24 представља број наставника који користе алате за проверу плагијаризма и број наставника који не користе ове алате приликом прегледа.

Табела 24 – Број испитаника који користе алате за ПП

Група	Број испитаника
Наставници који користе алате за ПП	46
Наставници који не користе алате за ПП	75

Табела 25 представља резултате добијене применом применом *Mann-Whitney U* теста.

Табела 25 – Резултати тестирања хипотезе X12

***U* статистика**	***p***	**Закључак**
1064.0	0.0283	Постоји статистички значајна разлика у значају проблема плагијата између две групе наставника.

Добијена *U* статистика износи 1064.0, што показује ранг разлике између две групе.

Добијена *p*-вредност је 0.0283, што је мање од нивоа значајности 0.05. Ово указује на то да постоји статистички значајна разлика у перцепцији значаја проблема плагијата између наставника који користе алате за проверу плагијата и оних који их не користе.

Резултати показују да наставници који користе алате за проверу плагијата имају значајно другачију перцепцију значаја проблема плагијата у поређењу са наставницима који их не користе. Овај резултат сугерише да коришћење алата за проверу плагијата може допринети подизању свести о проблему плагијата, што чини наставнике више усмереним ка превенцији и идентификацији плагијата у студентским радовима.

X13: Већина наставника сматра да је употреба алата за аутоматско оцењивање и проверу плагијата валидан и адекватан метод у процесу оцењивања студентских софтверских пројеката.

За хипотезу X13, примењен је хи-квадрат (χ^2) тест за испитивање повезаности између ставова наставника о адекватности метода за проверу квалитета кôда и плагијата (АМП) и њиховог коришћења алата за проверу плагијата (ОУАПП). Хи-квадрат (χ^2) тест независности омогућава утврђивање асоцијације између две категоријалне променљиве, а добијени резултати указују на статистички значајну повезаност између променљивих. Табела 26 представља резултате добијене применом применом хи-квадрат теста.

Табела 26 – Резултати тестирања хипотезе X13

Варијабле	χ^2 статистика	*p*	Степени слободе	Закључак
АМП и ОУАПП	11.42	0.0222	4	Постоји статистички значајна повезаност између две променљиве

Добијена χ^2 статистика износи 11.42, што указује на присуство асоцијације између ставова наставника о адекватности метода процене и њихове употребе алата за проверу плагијата.

Добијена *p*-вредност је 0.0222, што је мање од нивоа значајности 0.05. Ово указује на то да је повезаност статистички значајна, што значи да постоји мала вероватноћа да је ова веза настала случајно.

Резултати указују на то да постоји статистички значајна повезаност између ставова наставника о адекватности метода за проверу плагијата и квалитета кôда и њиховог коришћења алата за проверу плагијата. Овај резултат сугерише да наставници који сматрају овај метод адекватним имају већу вероватноћу да користе алате за проверу плагијата и квалитета кôда, што указује на важност алата у процесу евалуације студентских радова.

У наредном поглављу биће дискутовани представљени резултати.

5. Дискусија резултата истраживања

У оквиру овог поглавља представљена је анализа извођења истраживања и тумачење резултата истраживања у контексту значаја резултата у области статичке анализе кôда, као и теоријских и практичних импликација докторске дисертације. Ово поглавље организовано је у два дела, како би се представила дискусија резултата два истраживачка модела, који одговарају на засебна истраживачка питања.

5.1 Дискусија резултата истраживачког модела 1

Први део докторске дисертације је тежио да одговори на истраживачка питања: У којој мери фактори поставке пројекта утичу на квалитет кôда студентских пројеката? И друго питање, у којој мери квалитет кôда студентских пројеката утиче на академску успешност студената?

На основу представљених теоријских основа, и процедуре избора алата, дефинисан је истраживачки модел 1 (Слика 9), са постављених 9 хипотеза. На основу овако постављених хипотеза прикупљени су и обрађени подаци, који су описани у склопу поглавља 3.1. Добијени резултати спроведеног истраживања су описани у склопу поглавља 4.1. У наставку ће бити дискутовани резултати ИМ1 на основу којих ће бити креиране смернице за организацију и поставку софтверских пројеката које имају позитиван утицај на квалитет кôда.

5.1.1. Дискусија резултата испитивања хипотеза ИМ1

На основу примарних студија из спроведеног прегледа стања у области, издвојени су кључни фактори који могу утицати на квалитет кôда.

Приликом прикупљања софтверских пројеката над којима је спроведена анализа, услов укључивања пројеката подразумевао је равномерну расподелу свих типова пројеката према идентификованим кључним факторима.

У табели 27 приказан је удео пројеката према идентификованим факторима употребе различитих типова алата над софтверским пројектима у току развоја.

За вредности варијабли постављене су вредности 0 и 1, као што је описано у поглављу 3.1.2. Вредност 1 подразумева у варијаблама које се односе на употребу алата УСАК и УАВ да су на пројектима коришћени алати, док у случају варијабле НОН подразумева да је настава одржавана уживо. Коначно, у случају варијабле ТС, вредност 1 подразумева да је пројекат рађен тимски.

Табела 27 – Удео појављивања вредности бинарних варијабли

Фактор	1	0
Коришћени алати за статичку анализу кôда приликом рада на пројекту (УСАК)	23%	77%
Коришћени алати за верзионисање приликом рада на пројекту (УАВ)	74%	26%
Начин одржавања наставе (НОН)	87%	13%
Тип сарадње на пројекту (ТС)	67%	33%

Расподела и заступљеност вредности континуалних и ординалних метрика приказане су на сликама 11-19 у оквиру поглавља 3.1.2. Ове расподеле пружају детаљан увид у динамику софтверских пројеката обухваћених истраживањем. За сваки од седам посматраних фактора поставке пројеката обезбеђен је одговарајући број софтверских пројеката који припадају одговарајућој категорији, чиме је постигнута равномерна заступљеност свих категорија пројеката.

Основни скуп пројеката прикупљен је са различитих студијских програма на Факултету техничких наука, Универзитета у Новом Саду. Пројекти обухватају различите академске године, како би се осигурало да су покривене различите категорије и услови у којима су пројекти реализовани. На пример, пројекти из академске 2020/2021. године коришћени су за категоризацију онлајн наставе у оквиру варијабле НОН, чиме је омогућено анализирање специфичних аспеката рада у онлајн окружењу.

Над овако прикупљеним пројектима и информацијама о њиховој структури, коришћен је изабрани алат за статичку анализу кôда. Овај алат омогућио је анализу свих 500 софтверских пројеката, при чему су детаљно евидентиране метрике квалитета кôда за сваки пројекат. На основу прикупљених података, могуће је извршити дубљу анализу квалитета кôда и утврдити утицај различитих фактора на квалитет софтверских пројеката у академском окружењу.

Истраживачки модел 1 садржи кључне факторе поставке пројеката и хипотезе које претпостављају постојање везе између тих фактора односно варијабли истраживања. У наставку су дискутовани резултати испитаних хипотеза представљених у 4.1.1.

X1: Избор технологије има утицај на квалитет кôда софтверског пројекта.

Резултати истраживања потврђују значајну корелацију између избора технологије (ИТ) и квалитета кôда, што подржава предложену хипотезу. Анализа података открива да су пројекти реализовани са одређеним технологијама, као што су *.NET* и *Angular*, показали виши ниво квалитета кôда у поређењу са другим технологијама. Ово откриће указује на то да избор технологије не само да може директно утицати на резултате пројекта, већ такође представља важну компоненту у образовном контексту.

Одређене технологије, због своје структуре и алата које пружају, могу олакшати студентима усвајање добрих пракси кодирања. На пример, избор технологија које укључују алате за интеграцију са системима за верзионисање, статичку анализу кôда и аутоматско тестирање, не само да подиже ниво квалитета кôда, већ и омогућава студентима да развију вештине које су релевантне за индустрију. Укључивањем таквих технологија у образовни процес, предавачи могу усмерити студенте ка бољим праксама у развоју софтвера, што је кључно за њихово професионално напредовање.

Ипак, треба узети у обзир да све технологије нису једнако доступне свим студентима. Неке технологије, попут оних које захтевају дубље знање о архитектури и дизајну софтвера, могу представљати додатни изазов за почетнике. Програмски језици и алати који су повезани са сложенијим софтверским архитектурама могу захтевати више времена за учење, што заузврат може утицати на ниво квалитета кôда који студенти развијају. У оквиру образовног процеса, наведени фактори треба да буду узети у обзир како би се постигла равнотежа између подстицања учења кроз изазовне технологије и одржавања високог квалитета кôда.

Укључивање различитих технологија у наставни план и програм може обогатити образовно искуство студената и омогућити им да истраже предности и изазове различитих софтверских окружења. На тај начин се повећава њихова спремност за професионално окружење, где ће често бити суочени са избором технологија које треба да подрже специфичне захтеве пројеката. Дакле, увођење сложенијих технологија у оквиру студентских пројеката може утицати на развој њихових техничких и аналитичких вештина, омогућавајући им да разумеју како избор технологија утиче на квалитет и успех пројеката.

Резултати студије пружају значајну подршку хипотези X1, указујући на важност избора технологије у софтверским пројектима и њен директан утицај на квалитет кôда. У образовном контексту, ово откриће наглашава потребу за пажљивим избором технологија које ће не само подржати образовне циљеве, већ и допринети развоју вештина потребних за суочавање са комплексностима савременог софтверског инжењерства. На тај начин, избор технологија постаје кључна компонента у планирању наставних активности и утиче на будуће професионалне могућности студената.

X2: Тип сарадње на пројекту има утицај на квалитет кôда софтверског пројекта.

Резултати истраживања показују да тип сарадње (ТС), односно рад у тиму у односу на индивидуални рад, нема статистички значајан утицај на квалитет кôда (КК) у овом узорку. Међутим, ово откриће не искључује потенцијалне користи тимског рада у другим аспектима учења, као што су социјалне вештине и размена знања.

Тимски рад студентима пружа могућност да поделе одговорности и ангажују се у процесу ревизије кôда, што може допринети вишем нивоу квалитета. У тимском окружењу, студенти имају прилику да анализирају и усвајају различите перспективе решавања проблема, што потенцијално доводи до развоја кôда бољег квалитета кроз размену знања и искустава. Ова врста колаборативног учења може допринети квалитету кôда кроз структуиране дискусије о решењима и развоју критичког размишљања о исправности и одрживости кôда.

Упркос потенцијалним предностима, тимски рад може представљати изазове у виду неравномерне расподеле обавеза, што може утицати на квалитет кôда. Често се дешава да неки чланови тима преузимају већину одговорности, док други доприносе минимално. Овакве ситуације могу довести до смањења укупног квалитета кôда, јер су поједини делови пројекта израђени без довољно пажње или знања. Такође, недостатак јасно дефинисаних одговорности и ефективне комуникације у тиму може резултирати пропуштеним корацима у провери и тестирању кôда, што је критично за одржавање високог квалитета.

У оквиру истраживања, тимска сарадња се није показала као фактор који значајно утиче на квалитет кôда, што сугерише да одлука о формирању тимова или индивидуалном раду не мора имати директан ефекат на квалитет кôда у образовном контексту. Ипак, тимски пројекти имају значајне образовне предности у развоју меких вештина и припреми студената за рад у професионалним тимовима, што је важна компонента њиховог свеукупног образовног искуства.

X3: Величина пројекта утиче на квалитет кôда.

Резултати истраживања показују да постоји значајна негативна корелација између величине пројекта (ВП) и квалитета кôда, што подржава хипотезу да већи пројекти имају тенденцију да производе кôд нижег квалитета. Овај резултат указује на то да се са повећањем броја линија кôда јављају изазови у одржавању конзистентности, структуре и квалитета кôда, што је од значаја за развој софтверског инжењерства.

Како пројекти постају већи, сложеност кôда се повећава, што повећава вероватноћу грешака и неправилности у кодирању. Већи пројекти често захтевају више времена за тестирање и ревизију, што може да доведе до пропуштања неких аспеката квалитета кôда. Са повећањем величине пројекта, одржавање јединствених стандарда постаје изазов, посебно ако постоји потреба за чешћим ажурирањима или изменама. Стога, већи пројекти захтевају ефикаснију примену алата за статичку анализу кôда и строже праћење стандарда за управљање сложеношћу кôда.

Наставници могу да се фокусирају на подучавање студената техникама модуларизације и структурирања кôда како би се успешно управљало сложеним пројектима. Коришћење алата за верзионисање и континуирану интеграцију такође може помоћи студентима да одрже висок квалитет кôда, чак и када рад на пројекту постане обиман. Додатно, расподела пројекта на мање, независне модуле може олакшати управљање и омогућити студентима да се усредсреде на контролу квалитета појединих компоненти.

Са становишта образовања, пројекти који су већи могу помоћи студентима да разумеју и примене принципе управљања софтверским пројектима у реалним условима. Укључивање великих пројеката у наставу омогућава студентима да стекну увид у изазове који произилазе из рада на обимним софтверским системима. Ипак, потребно је осигурати да студенти имају приступ потребним ресурсима и алатима како би одржали квалитет кôда и стекли искуство које је релевантно за будуће радно окружење.

Ова студија подржава хипотезу да величина пројекта утиче на квалитет кôда, што указује на то да се са повећањем обима кôда јављају додатни изазови у одржавању квалитета. Овај налаз наглашава потребу за увођењем пракси и алата који ће помоћи студентима да се суоче са овим изазовима, као што су модуларизација, коришћење алата за контролу квалитета и редовна анализа кôда. Применом наведених стратегија, образовне институције могу припремити студенте да управљају и одржавају квалитет кôда чак и у великим и сложеним софтверским пројектима.

X4: Академска година студента позитивно утиче на квалитет развијеног кôда.

Ова студија подржава хипотезу да величина пројекта утиче на квалитет кôда, што указује на то да се са повећањем обима кôда јављају додатни изазови у одржавању квалитета. Овај налаз наглашава потребу за увођењем пракси и алата који ће помоћи студентима да се суоче са овим изазовима, као што су модуларизација, коришћење алата за контролу квалитета и редовна анализа кôда. Применом наведених стратегија, образовне институције могу припремити студенте да управљају и одржавају квалитет кôда чак и у великим и сложеним софтверским пројектима.

Резултати истраживања указују на то да академска година студента (АГ), као индикатор искуства и нивоа знања, нема значајан утицај на квалитет развијеног кôда. Ово сугерише да су други фактори, попут технологије и статичке анализе, можда важнији у одређивању квалитета кôда у односу на саму годину студија студента. Ипак, академска година може имати индиректан утицај на квалитет кôда кроз повећано искуство и познавање софтверских алата и техника.

Са сваком вишом академском годином, очекује се да студенти стичу дубље разумевање принципа софтверског инжењерства, обухватајући теме као што су структура кôда, дизајн софтвера и напредније технике програмирања. Виши нивои образовања често укључују и боље разумевање концепата, што може потенцијално допринети вишем квалитету кôда, у поређењу са студентима нижих година. Иако ово истраживање није показало директну корелацију између академске године и квалитета кôда, повећање искуства и познавање алата може позитивно утицати на способност студента да развије квалитетан кôд.

Студенти који су на нижим годинама студија могу имати ограничено искуство у примени софтверских принципа и алата за анализу кôда. То може резултирати кôдом нижег квалитета у поређењу са студентима на вишим годинама студија, који су упознати са напреднијим техникама и алатима за проверу и побољшање кôда. Овај недостатак искуства може утицати на њихову способност да примене сложеније технике кодирања и доследно прате стандарде квалитета.

Узимајући у обзир ове факторе, едукатори могу прилагодити наставне планове и програме како би обезбедили постепено повећање нивоа сложености пројеката са академском годином студената. Укључивање додатних радионица и менторства за студенте нижих година може им помоћи да развију вештине потребне за одржавање квалитета кôда. Са друге стране, студентима виших година, сложени пројекти могу бити изазов, који им омогућава да примене свеобухватно знање и побољшају своје аналитичке способности у решавању софтверских проблема.

Иако академска година није показала значајну корелацију са квалитетом кôда у овој студији, она остаје важан аспект који утиче на искуство и вештине студената. Повећање знања и примене алата кроз године студија потенцијално доприноси бољем разумевању софтверског инжењерства, што може индиректно побољшати квалитет кôда. Прогресивно излагање студената вештинама и алатима током академских година може побољшати њихову способност да развијају кôд високог квалитета и успешно одговоре на изазове савременог софтверског развоја.

X5: Употреба алата за верзионисање позитивно утиче на квалитет кôда.

Резултати истраживања нису показали статистички значајан утицај употребе алата за верзионисање (VAB) на квалитет кôда у оквиру овог узорка. Међутим, алати за верзионисање, имају значајну улогу у професионалном развоју софтвера и подстицању добрих пракси кодирања. Овај налаз указује на могућност да студенти, посебно они на нижим годинама студија, још увек не користе пуни потенцијал алата на начин који би значајно утицао на квалитет њиховог кôда.

Алат за верзионисање омогућава праћење промена у кôду, олакшавајући повратак на претходне верзије и контролу над развојем пројекта. У професионалном окружењу, алати играју кључну улогу у колаборативном раду, јер омогућавају истовремену сарадњу више програмера без ризика од преписивања или губитка кôда. Употреба алата пружа студентима могућност да развију боље праксе у управљању кôдом, укључујући документовање промена, што може допринети квалитету кôда кроз унапређење структуре и прегледност.

Недовољно искуство са напредним функцијама, као што су гранање, спајање и ревизија кôда, може ограничити потенцијални утицај алата за верзионисање на квалитет кôда. Да би студенти стекли потпунији увид у предности алата, потребно је увести додатну обуку и праксу која наглашава важност управљања кôдом и примени свих аспеката верзионисања.

Образовне институције могу подстицати студенте на коришћење алата за верзионисање кроз тимске пројекте и пројекте који захтевају честа ажурирања кôда. Студенти могу добити задатке који захтевају употребу гранања и спајања кôда, што би их подстакло да се упознају са напреднијим функцијама алата за верзионисање. Такође, могуће је организовати радионице које би пружиле практичне савете за ефикасно коришћење ових алата, што би допринело њиховој спремности за рад у професионалном окружењу где је квалитет кôда тесно повезан са ефикасном контролом верзија.

Иако употреба алата за верзионисање није показала значајан директан утицај на квалитет кôда у овом истраживању, њихова важност у управљању развојем софтвера је неоспорна. Са већим искуством и напреднијим коришћењем алата, студенти могу остварити бољу контролу над квалитетом кôда, што је од значаја за њихово професионално усавршавање. Препорука за образовне установе је да укључе систематску обуку за употребу алата за верзионисање како би се унапредиле вештине студената у управљању кôдом и одржавању његовог квалитета, што ће бити од користи за будућу професионалну праксу студената у софтверском инжењерству. Поред тога, алати за верзионисање често омогућавају и интеграцију са статичком анализом кôда, чиме се може обезбедити правовремено откривање грешака и континуирано мерење квалитета сваке верзије.

Х6: Употреба алата за статичку анализу кôда позитивно утиче на квалитет кôда.

Резултати истраживања потврђују значајну позитивну корелацију између употребе алата за статичку анализу кôда (УСАК) и квалитета кôда, што подржава хипотезу да алати доприносе већем квалитету кôда у студентским пројектима. Ово откриће указује на то да алати за статичку анализу, као што су *SonarQube* и *PMD*, могу бити изузетно корисни у образовном контексту, јер омогућавају студентима да идентификују и исправе грешке пре него што кôд уђе у фазу тестирања или продукције.

Алат за статичку анализу кôда омогућава аутоматско откривање грешака, рањивости и одступања од квалитетних стандарда током процеса развоја кôда. Алати пружају студентима повратне информације о њиховом кôду и уводе их у концепте као што су сигурност, одрживост и оптимизација. Употребом алата за статичку анализу, студенти могу научити како да избегну типичне грешке у кодирању и стекну вештине потребне за развој квалитетног и поузданог софтвера.

Иако су алати корисни, почетни рад са њима може бити изазован за студенте који немају претходно искуство у њиховој употреби. Алати за статичку анализу често генеришу велики број упозорења и извештаја, што може бити преоптерећујуће за почетнике. Да би се избегла преоптерећеност, наставници могу постепено уводити ове алате у наставни план и програм, почевши са основним функцијама. Како студенти напредују у знању и вештинама, могу се уводити и напреднији аспекти.

Образовне институције могу укључити алате за статичку анализу кôда у пројектне задатке како би подстакле студенте да одржавају висок квалитет кôда током развоја софтвера. Радионице и практични задаци који захтевају употребу алата могу помоћи студентима да стекну конкретно знање о томе како идентификовати и решавати проблеме у кôду. Кроз овакве активности, студенти не само да унапређују свој технички капацитет, већ и развијају критичко размишљање потребно за анализу и ревизију кôда, што их припрема за реалне изазове у софтверској индустрији.

Ова студија потврђује да употреба алата за статичку анализу кôда позитивно утиче на квалитет кôда, што наглашава њихову вредност у образовању будућих софтверских инжењера. Примена алата у образовном контексту омогућава студентима да стекну практичне вештине у процени и побољшању кôда, што је кључно за одржавање високих стандарда у развоју софтвера. Едукатори могу користити ове алате као средство за

унапређење наставног плана и програма, истовремено припремајући студенте за изазове и захтеве савременог софтверског инжењерства.

X7: Начин одржавања наставе утиче на квалитет кôда.

Резултати истраживања показали су да начин одржавања наставе (НОН), односно да ли се настава одвија онлајн или у учионици, нема значајан утицај на квалитет кôда који студенти развијају. Ово сугерише да су други фактори, као што су употреба алата за статичку анализу и избор технологије, можда важнији у одређивању квалитета кôда у студентским пројектима. Ипак, начин одржавања наставе може имати индиректан утицај на друге аспекте учења, као што су доступност ресурса и квалитет интеракције са инструкторима.

Традиционална настава омогућава студентима непосредну интеракцију са инструкторима и колегама, што може подстаћи брже решавање проблема и разјашњавање нејасноћа у вези са развојем кôда. Са друге стране, онлајн настава пружа студентима већу флексибилност и могућност да приступе наставним материјалима по потреби, што може бити корисно за самостално учење. Међутим, ова студија показује да сам начин одржавања наставе не утиче директно на квалитет кôда, што указује на то да и онлајн и традиционална настава могу бити подједнако ефикасне у преношењу техничког знања потребног за развој софтвера.

Онлајн настава може донети изазове у виду мање директне подршке и интеракције, што може довести до тога да студенти осећају мање вођства током рада на пројектима. Насупрот томе, традиционална настава може бити ограничена када је у питању приступ алатима и ресурсима који су доступни онлајн. У оба случаја, кључно је обезбедити студентима приступ алатима за учење и додатну подршку, без обзира на начин наставе. Тиме се осигурава да студенти имају све што је потребно за развој квалитетног кôда, без обзира на формат наставе.

Образовне институције могу интегрисати хибридни приступ који комбинује предности оба модела наставе, омогућавајући студентима приступ различитим ресурсима у учионици и онлајн. Примена интерактивних алата за подршку учењу, као што су онлајн форуми, виртуелне учионице и алати за дељење кôда, може помоћи студентима да имају подршку и у традиционалној и у онлајн настави. Образовне институције могу такође организовати редовне консултације и менторске сесије како би студентима пружили подршку у решавању проблема и унапређењу квалитета кôда.

Ово истраживање указује да начин одржавања наставе, било да је онлајн или традиционалан, нема директан утицај на квалитет кôда који студенти производе. Међутим, оба модела имају своје предности и изазове, и могу бити подједнако ефикасни у развоју техничких вештина код студената ако се обезбеде одговарајући ресурси и подршка. Препорука за образовне установе је да развију наставне планове који укључују најбоље праксе из оба приступа, омогућавајући студентима флексибилност и подршку која је потребна за одржавање квалитета кôда у свим условима наставе.

X8: Квалитет кôда је у позитивној корелацији са оценом студента.

Резултати истраживања показују да постоји значајна позитивна корелација између квалитета кôда (КК) и оцена које студенти добијају. Овај налаз указује на то да студенти који развијају кôд високог квалитета обично постижу и боље оцене, што подржава хипотезу да су техничке вештине кључне за академски успех у областима софтверског инжењерства. Корелација између квалитета кôда и академског успеха наглашава важност фокуса на квалитет у образовању, како би се осигурало да студенти разумеју и примењују принципе доброг кодирања.

Студенти који се придржавају најбољих пракси кодирања, као што су добра структура кôда, избегавање грешака и поштовање стандарда квалитета, могу лакше добити боље оцене, јер њихов кôд рефлектује дубље разумевање концепата програмирања. Добар квалитет кôда такође показује да су студенти способни да примењују научене принципе у решавању проблема и да имају вештине потребне за анализу и ревизију свог рада, што представља аспекте који су важни за академске оцене. На овај начин, постоји јасна веза између техничког квалитета производа и перформанси студента у академском контексту.

Наставници могу користити резултате анализе квалитета кôда као мерило за оцену студентског знања и напретка. Коришћењем алата за статичку анализу кôда и сличних мера, наставници могу објективно оцењивати квалитет пројеката и повезати оцене са показатељима техничког знања. Додатно, укључивање алата за анализу кôда у настави може подстаћи студенте да обрате више пажње на техничке аспекте својих пројеката, што ће позитивно утицати на њихове оцене. Осим тога, редовне повратне информације о квалитету кôда могу помоћи студентима да побољшају своје вештине током студија, унапређујући, не само своје академске резултате, већ и припрему за професионално окружење.

Студија подржава хипотезу да је квалитет кôда позитивно повезан са оценама студената, што указује на то да студенти који производе кôд високог квалитета често постижу боље академске резултате. Овај налаз наглашава важност развоја наставних стратегија које истичу значај квалитета кôда и његов утицај на академски успех. Едукатори би требало да наставе са промоцијом најбољих пракси у кодирању и пружањем објективних мера квалитета као дела наставног процеса, чиме ће осигурати да студенти разумеју важност техничких аспеката кодирања и примене ове вештине за постизање виших оцена.

Х9: Квалитет кôда је у позитивној корелацији са бројем рокова потребним за завршетак пројекта.

Резултати истраживања показују да време потребно за завршетак пројекта (БРП) нема значајан утицај на квалитет кôда (КК) у овом узорку. Овај налаз сугерише да, упркос очекивањима, дуже време посвећено раду на пројекту не гарантује нужно квалитетнији кôд. Ово може указивати на то да је утицај времена проведеног на пројекту комплекснији него што се претпоставља и да други фактори, као што су искуство студента и алати који се користе, играју важнију улогу у дефинисању квалитета кôда.

Студенти који проводе више времена на пројектима могу бити склонији детаљном прегледу и побољшању свог кôда. Међутим, истраживање показује да, без обзира на дужину времена, студенти који немају одговарајуће техничке вештине или подршку, могу доживети смањен квалитет кôда. Са друге стране, краћи временски оквири могу подстаћи брже одлуке и смањење прегледности и структуре кôда, што утиче на квалитет. Добијени налази указују на то да само време посвећено раду на пројекту можда није довољан фактор да се обезбеди висок квалитет кôда.

За унапређење квалитета кôда, наставници могу охрабрити студенте да се фокусирају на стратегије управљања временом и ефикасне праксе развоја кôда, а не само на продужење времена рада на пројекту. Интегрисање обавезних рокова и периодичних прегледа пројеката током рада може помоћи студентима да постигну бољи квалитет кôда без ослањања искључиво на време. Образовне институције могу такође да обезбеде додатну обуку и ресурсе који ће студентима помоћи да разумеју како да ефективно управљају својим временом и развијају кôд који испуњава квалитативне стандарде.

Ово истраживање указује да време потребно за завршетак пројекта није директно повезано са квалитетом кôда. Овај налаз наглашава важност других фактора, као што су употреба алата и вештине студената, који имају значајнији утицај на квалитет кôда.

Наставници и студенти се могу фокусирати на побољшање техничких аспеката и организационих вештина уместо на дужину времена које је потребно за завршетак пројекта, како би се обезбедио конзистентан квалитет кôда у различитим временским оквирима.

5.1.2. Дискусија валидације резултата

У оквиру овог поглавља представљена је анализа резултата валидационог истраживања и њихово поређење са резултатима из иницијалног истраживања. Циљ је да се утврди да ли се налази из првобитног истраживања могу потврдити на другом узорку и да се дискутују могуће разлике и сличности.

Овај део докторске дисертације фокусиран је на валидацију резултата добијених из истраживачког модела 1 (ИМ1), где су основна истраживачка питања:

1. Да ли се резултати из иницијалног истраживања могу потврдити на другом, независном узорку?
2. Које су сличности и разлике између резултата из два истраживања?

Валидационо истраживање је спроведено на узорку од 210 софтверских пројеката са Факултета електротехнике и рачунарства, Универзитета у Загребу. Због ограничења у доступности података, фактори, односно хипотезе X5, X6, X7 и X9 из иницијалног модела нису могли бити анализирани. Остали фактори су прикупљени и обрађени на исти начин као у иницијалном истраживању.

У наставку ће бити дискутовани резултати испитивања хипотеза X1 – X4 и X8 у оквиру валидационог истраживања и њихово поређење са резултатима из иницијалног истраживања.

X1: Избор технологије има утицај на квалитет кôда софтверског пројекта.

Резултати валидационог истраживања потврђују значајну корелацију између избора технологије (ИТ) и квалитета кôда (КК), што је у складу са налазима из иницијалног истраживања. Ово указује на то да је избор технологије важан фактор који утиче на квалитет кôда, без обзира на различите академске институције и околности пројеката.

Поновљена потврда ове хипотезе на другом узорку сугерише да одређене технологије, због својих карактеристика и доступних алата, омогућавају студентима да лакше примене добре праксе кодирања. То потврђује значај пажљивог одабира технологија у образовном контексту, како би се подстакао развој квалитетног кôда.

X2: Тип сарадње на пројекту има утицај на квалитет кôда софтверског пројекта.

У валидационом истраживању, резултати показују да тип сарадње (ТС) има значајан утицај на квалитет кôда, што је супротно налазима из иницијалног истраживања, где овај фактор није показао статистички значајан утицај.

Ова разлика може бити последица различитих начина организовања тимског рада у две институције. Могуће је да су студенти на Универзитету у Загребу имали бољу структуру тимског рада, бољу подршку у колаборацији или су били подстакнути на ефикаснију сарадњу, што је резултирало вишим квалитетом кôда у тимским пројектима.

Ово откриће сугерише да тип сарадње може имати значајан утицај на квалитет кôда, али да тај утицај може зависити од контекста и начина на који се тимски рад организује и подржава.

X3: Величина пројекта утиче на квалитет кôда.

Резултати валидационог истраживања показују да величина пројекта (ВП) нема статистички значајан утицај на квалитет кôда, што се разликује од налаза из иницијалног истраживања, где је постојала негативна корелација између величине пројекта и квалитета кôда.

Ово може указивати на то да су студенти у валидационом узорку били боље припремљени да управљају већим пројектима или су користили алате и технике које су им помогле да одрже квалитет кôда без обзира на величину пројекта. Такође, мања је разлика у обиму пројеката у валидационом узорку, што је смањило варијацију и утицај величине на квалитет кôда.

Ова разлика наглашава потребу за даљим истраживањем како би се разумели фактори који могу ублажити негативан утицај величине пројекта на квалитет кôда.

X4: Академска година студента позитивно утиче на квалитет развијеног кôда.

Резултати валидационог истраживања потврђују да академска година (АГ) има значајан утицај на квалитет кôда, што је у супротности са налазима из иницијалног истраживања, где овај фактор није показао значајан утицај.

Ово сугерише да на Универзитету у Загребу постоји јаснији напредак у вештинама кодирања са напредовањем кроз академске године. Студенти виших година можда имају више искуства и знања која им омогућавају да развију квалитетнији кôд.

Ова разлика указује на то да академска година може бити важан фактор утицаја на квалитет кôда, али да то може зависити од наставног програма и начина на који се вештине развијају током студија.

X8: Квалитет кôда је у позитивној корелацији са оценом студента.

Као и у иницијалном истраживању, резултати валидационог истраживања потврђују значајну позитивну корелацију између квалитета кôда и оцене коју студент добија на предмету. Ово конзистентно откриће наглашава да је квалитет кôда добар предиктор академске успешности и да студенти који производе квалитетан кôд добијају боље оцене.

Овај налаз потврђује важност фокуса на квалитет кôда у настави и оцењивању, јер директно утиче на академски успех студената.

У претходном поглављу показано је да постоји јасна корелација између квалитета кôда и оцене коју студент добија на предмету. Од 210 студената, 42 студента је развило висококвалитетан кôд и добила високу оцену. Да би се утврдило да ли се ова хипотеза може генерализовати на општи узорак, спроведена је додатна статистичка анализа користећи Бернулијеву расподелу и кумулативну дистрибутивну функцију.

Испитана је вероватноћа да ће се тачно 42 рада високог квалитета (што је 20% од 210) наћи у узорку исте величине. Добијени резултат је 6,87%, што је највећа вероватноћа за овај број успешних исхода. Ово је у складу са очекивањем на основу Бернулијеве расподеле, што указује на стабилност вероватноће појаве радова високог квалитета у узорцима ове величине.

Даље је анализирана кумулативна дистрибутивна функција за различите опсеге броја радова високог квалитета:

- Опсег од 25 до 45,
- Опсег од 30 до 50 и
- Опсег од 35 до 60.

Кумулативна дистрибутивна функција, приказана на слици 30, показује вероватноћу да ће број радова високог квалитета бити мањи од одређене, или једнак одређеној вредности. На тај начин, показана је постепена акумулација вероватноће, што потврђује да је дистрибуција стабилна и омогућава интерполацију на било који узорак сличне величине.

Ова статистичка анализа додатно подржава хипотезу X8, јер показује да постоји стабилна и предвидива вероватноћа да ће студенти који производе висококвалитетан кôд добити и високе оцене. То указује на то да се квалитетом кôда може успешно предвидети оцена студента, што је значајно за процес оцењивања у академским установама.

Резултати валидационог истраживања пружају додатну подршку налазима из иницијалног истраживања, али исто тако указују на разлике које треба детаљније истражити. Потврда утицаја избора технологије и корелације између квалитета кôда и оцене студента наглашавају важност описаних фактора у образовном контексту.

Разлике утицаја типа сарадње, величине пројекта и академске године указују на потребу за прилагођавањем наставних метода и подршке студентима у складу са специфичностима сваке институције. Препоручује се да се додатна пажња посвети организацији тимског рада и развоју вештина кодирања кроз академске године.

Коначно, резултати сугеришу да је важно континуирано пратити и анализирати факторе који утичу на квалитет кôда и академски успех студената, како би се унапредили наставни процеси и исходи учења.

5.1.3. Смернице за организацију и поставку софтверских пројеката

На основу резултата добијених у оквиру првог истраживачког модела, формулисане су смернице које имају за циљ да помогну образовним институцијама и наставницима у организацији и поставци софтверских пројеката на курсу. Ове смернице су осмишљене да унапреде квалитет кôда студентских пројеката и подстакну академски успех студената.

Опште смернице:

- Пажљив избор технологија: Резултати истраживања потврђују да избор технологије значајно утиче на квалитет кôда (X1). Наставници треба да бирају технологије које подстичу добре праксе кодирања и нуде интеграцију са алатима за контролу квалитета. Предност треба дати технологијама са јаком стручном заједницом и обилном документацијом.
- Интеграција алата за статичку анализу кôда: Употреба алата за статичку анализу кôда позитивно утиче на квалитет кôда (X6). Алати помажу студентима да идентификују и исправе грешке у раној фази развоја. Наставници треба да укључе ове алате у наставни план и обезбеде обуку за њихово коришћење.
- Управљање величином и сложеношћу пројеката: Већи пројекти могу негативно утицати на квалитет кôда (X3). Наставници треба да подучавају технике модуларизације и структурирања кôда, како би студенти могли ефикасно да управљају сложенијим пројектима.
- Фокус на квалитет кôда у оцењивању: Квалитет кôда је позитивно повезан са оценама студената (X8). Укључивање метрика квалитета кôда у критеријуме оцењивања подстиче студенте да посвете већу пажњу писању квалитетног кôда.
- Подршка тимском раду: Утицај тимског рада на квалитет кôда може варирати (X2). Организација тимских пројеката са јасним смерницама и подршком може побољшати квалитет кôда и развој меких вештина.

Кораци за организацију и поставку пројеката:

- Избор одговарајућих технологија.
  - Анализирати доступне технологије и одабрати оне које подстичу добре праксе кодирања.
  - Предност дати технологијама које се добро интегришу са алатима за статичку анализу кôда и контролу верзија.
- Интеграција алата за статичку анализу у наставу.
  - Увести алате као што су *SonarQube* и *PMD* од раних фаза курса.
  - Организовати радионице за практичну обуку студената у коришћењу алата.
- Подучавање техника управљања сложеношћу.
  - Укључити теме о модуларном програмирању и управљању сложеношћу у наставни план.
  - Доделити пројекте који захтевају примену алата за одржавање квалитета кôда.
- Интеграција квалитета кôда у систем оцењивања.
  - Дефинисати јасне критеријуме који укључују метрике квалитета кôда.
  - Пружити студентима повратне информације и смернице за унапређење кôда.
- Обука и подстицање коришћења алата за верзионисање.
  - Укључити вежбе које захтевају употребу система за контролу верзија као што је *Git.*
  - Подстаћи коришћење напредних функција као што су гранање и спајање.
- Ефективна организација тимског рада.
  - Обезбедити смернице за поделу задатака и комуникацију у тиму.
  - Пратити напредак и допринос сваког члана тима.
- Комбинација наставних метода.
  - Искористити предности и онлајн и традиционалне наставе.
  - Обезбедити доступност ресурса и подршке без обзира на начин одржавања наставе.

Пажљивом применом навених корака и смерница, образовне институције и наставници би значајно унапредили квалитет наставе у области софтверског инжењерства. Студенти би имали прилику да раде са одговарајућим технологијама и алатима који промовишу добре праксе кодирања, што би им омогућило да развијају кôд високог квалитета. Интеграција алата за статичку анализу, ефикасно управљање сложеношћу пројеката и фокус на квалитет кôда у оцењивању би допринели дубљем разумевању и усвајању важних концепата. Подршка тимском раду и коришћење савремених система за контролу верзија би такође унапредили међусобну сарадњу и техничке вештине студената. Кроз ове мере, студенти би постали способнији да генеришу висококвалитетан кôд, што би у крајњој линији довело до већег академског успеха и припремило их за изазове у професионалној каријери у софтверском инжењерству.

5.2 Дискусија резултата истраживачког модела 2

Други део докторске дисертације је тежио да одговори на истраживачко питање: Како наставници оцењују важност и потребу за алатима за проверу плагијата и квалитета кôда у прегледу софтверских пројеката?

На основу представљених теоријских основа и прегледа стања у области, дефинисан је истраживачки модел 2 (слика 10), са постављене 4 хипотезе. На основу овако постављених хипотеза прикупљени су и обрађени подаци описани у склопу поглаља 3.2, и затим су добијени резултати спроведеног истраживања описани у склопу поглавља 4.2. У наставку ће бити дискутовани резултати у склопу ИМ2 на основу којих ће бити креиране смернице за организацију оцењивања софтверских пројеката.

5.2.1. Дискусија резултата испитивања хипотезе ИМ2

Истраживачки модел 2 садржи кључне факторе који укључују плагијаризам и аутоматско оцењивање софтверских пројеката заснованих на статичкој анализи кôда. У наставку су дискутовани резултати испитаних хипотеза представљених у 4.2.

Х10: Наставници који сматрају проблем плагијата значајним показују већу спремност за коришћење алата за проверу плагијата и процену квалитета кôда.

Резултати истраживања показују да постоји позитивна корелација средње јачине између значаја проблема плагијата (ПП) и спремности наставника за коришћење алата (СНКА). Спирманов коефицијент корелације износио је 0.333, а *p*-вредност је износила 0.0013, што указује на статистички значајну повезаност између променљивих. Овај налаз сугерише да наставници, који проблем плагијата сматрају значајним, показују већу спремност за употребом алата који олакшавају детекцију и процену квалитета кôда.

Овај резултат је у складу са очекивањима, јер би се могло претпоставити да наставници који придају велику важност проблему плагијата настоје да користе додатне алате како би побољшали процес оцењивања и осигурали академски интегритет. Употреба алата за проверу плагијата и квалитета кôда омогућава наставницима да ефикасније идентификују и управљају потенцијалним случајевима неакадемског понашања код студената.

Међутим, иако је корелација статистички значајна, јачина везе указује на то да постоје и други фактори који могу утицати на спремност наставника за коришћење алата. На пример, недовољна обука или техничка подршка, као и перцепција сложености коришћења алата, могу играти значајну улогу у одлуци наставника да имплементирају ове алате у своју наставну праксу.

Ови налази директно доприносе истраживачком питању (ИП3) које испитује како наставници оцењују важност и потребу за алатима за проверу плагијаризма и квалитета кôда у току прегледања софтверских пројеката. Резултати сугеришу да наставници који сматрају плагијаризам значајним проблемом показују већу спремност за коришћење алата, што указује на то да је перцепција важности проблема круцијалан фактор у прихватању и употреби технологије у едукативном процесу. Ово истраживање имплицира да образовне институције треба да повећају свест о проблему плагијата и обезбеде ресурсе за лакше усвајање алата који могу унапредити процес евалуације студентских радова.

Х11: Наставници, који сматрају преглед великог броја студентских пројеката без алата изазовним, показују већу спремност за коришћење алата за проверу плагијата и процену квалитета кôда.

Резултати истраживања показују да постоји позитивна корелација средње јачине између изазовности прегледа великог броја пројеката без алата (ИПБА) и спремности наставника за коришћење алата (СНКА). Спирманов коефицијент корелације износио је 0.344, а p-вредност је 0.0009, што указује на статистички значајну повезаност између промењљивих. Овај налаз сугерише да наставници, који сматрају да је преглед великог броја пројеката без употребе алата изазован, показују већу спремност за коришћење алата који могу олакшати процес процене и прегледа.

Овај резултат је у складу са претпоставком да наставници, који се суочавају са већим оптерећењем током прегледа пројеката, теже ка употреби технолошких решења која могу смањити време и напор потребан за процену. Алати за проверу плагијата и квалитета кôда могу значајно помоћи наставницима да брже идентификују потенцијалне проблеме и обезбеде објективније оцењивање.

Иако је корелација статистички значајна, резултати такође указују на то да постоје и други фактори који могу утицати на спремност наставника да користе ове алате. То могу бити техничке баријере, као и недовољна свест о могућностима и предностима које алати нуде.

Ови налази доприносе одговору на истраживачко питање о процени важности и потребе за алатима за проверу плагијаризма и квалитета кôда у прегледу софтверских пројеката. Резултати сугеришу да изазовност прегледа без алата има значајну улогу у спремности наставника за њихову употребу. Ово указује на то да, уколико наставници сматрају процес прегледа изазовним, постоји већа вероватноћа да ће прихватити технологију која може олакшати овај процес. Образовне институције могу користити овај налаз као основ за увођење обуке и подршке за коришћење алата, како би се наставници осећали сигурније и мотивисаније да их примене у свом раду.

X12: Наставници који користе алате за проверу плагијата сматрају да је проблем плагијата значајнији у односу на наставнике који не користе те алате.

Резултати *Mann-Whitney U* теста показали су да постоји статистички значајна разлика у перцепцији значаја проблема плагијата (ПП) између наставника који користе алате за проверу плагијата и оних који их не користе. U статистика износила је 1064.0, а p-вредност је била 0.0283, што је мање од нивоа значајности од 0.05. Ово указује на то да наставници који користе алате за проверу плагијата придају већу важност проблему плагијата у односу на оне који их не користе.

Овај налаз је у складу са претпоставком да су наставници који активно користе алате за проверу плагијата свеснији проблема и могућих последица плагијата, што их мотивише да се додатно ослоне на технолошка решења за одржавање академског интегритета. Алати за проверу плагијата могу помоћи наставницима да брже и прецизније идентификују сумњиве активности, што их чини вредним у наставној пракси.

Иако је *Mann-Whitney U* тест показао статистички значајну разлику (p-вредност $<$ 0.05), величина ефекта је мала ($r \approx 0.1$). То значи да иако постоји разлика у перцепцији значаја проблема плагијата између наставника који користе алате и оних који их не користе, та разлика није јака. Овај налаз сугерише да иако коришћење алата може утицати на перцепцију плагијата, тај утицај није изразито јак и могући су други фактори који играју значајнију улогу у формирању ставова наставника.

Импликације за истраживачко питање: Овај резултат указује на то да, иако употреба алата може имати одређени утицај на свест о значају плагијата, тај утицај је релативно слаб. Образовне институције би могле да истраже и друге факторе који утичу на перцепцију плагијата, као што су обука и подршка за коришћење алата, искуство наставника и њихова свест о проблему.

X13: Већина наставника сматра да је употреба алата за аутоматско оцењивање и проверу плагијата валидан и адекватан метод у процесу оцењивања студентских софтверских пројеката.

Резултати хи-квадрат (χ^2) теста независности показали су да постоји статистички значајна повезаност између ставова наставника о адекватности метода за проверу плагијата и квалитета кôда (АМП) и њиховог коришћења алата за проверу плагијата (ОУАПП). Добијена χ^2 статистика износила је 11.42, а *p*-вредност је 0.0222, што је мање од нивоа значајности од 0.05. Добијени резултати указују на то да је повезаност између двеју променљивих статистички значајна.

Овај налаз потврђује да наставници, који методе за проверу квалитета кôда и плагијата сматрају адекватним, чешће користе алате за њихову примену. Ово сугерише да наставници, који имају позитиван став према коришћењу алата, виде вредност и корисност алата у свом наставном процесу, што их мотивише да их интегришу у свој рад.

Да би била утврђена јачина повезаности између категоријалних променљивих коришћен је *Cramér's V* коефицијент, овај коефицијент показује колико је значајна веза између две променљиве у хи-квадрат (χ^2) тесту. Вредност *Cramér's V* указује на то колико је јака повезаност између ставова наставника о адекватности метода са једне стране и њихове употребе алата за проверу плагијата са друге стране, при чему већа вредност означава јачу везу.

Вредност *Cramér's V* од 0.338 указује на значајну повезаност између ставова наставника о адекватности алата и њиховог коришћења. То значи да су наставници који сматрају да су методе за проверу квалитета кôда и плагијата адекватне много вероватније да ће користити алате у свом раду.

Овај налаз значајно доприноси истраживачком питању које испитује како наставници оцењују важност и потребу за алатима за проверу плагијата и квалитета кôда. Снажан ефекат указује на то да перцепција наставника о ефикасности метода игра кључну улогу у њиховом прихватању и примени у настави. Ово може значити да би образовне институције требало да промовишу и подстичу позитивну перцепцију кроз обуке, едукацију и демонстрације делотворности алата. На тај начин би се повећала њихова употреба међу наставницима, што би унапредило процес оцењивања студентских пројеката и одржавање академског интегритета.

5.2.2. Смернице за организацију процеса оцењивања софтверских пројеката

На основу резултата и дискусије у оквиру истраживачког модела 2 (ИМ2), могу се формулисати конкретне смернице за организацију процеса оцењивања студентских софтверских пројеката. Ове смернице су осмишљене тако да унапреде ефикасност, објективност и интегритет процеса оцењивања, узимајући у обзир ставове и спремност наставника за коришћење алата за проверу плагијата и квалитета кôда.

1. корак: Провера плагијата у кôду пројекта помоћу алата за статичку анализу кôда

Први корак након предаје пројекта подразумева систематичну проверу плагијата у кôду студента. Наставник користи алате за статичку анализу кôда који су специјализовани за детекцију сличности и копираних делова кôда. Резултати истраживања показују да наставници који сматрају проблем плагијата значајним показују већу спремност за коришћење алата (X10 и X12). Ова пракса не само да осигурава академски интегритет, већ и подстиче студенте да развијају оригинална решења и унапређују своје вештине програмирања.

2. корак: Провера основних функционалности пројекта у односу на захтеве

Уколико пројекат успешно прође проверу плагијата, наставник приступа провери основних функционалности пројекта. Овај корак укључује тестирање да ли софтвер испуњава задате спецификације и функционалне захтеве. Резултати истраживања сугеришу да наставници који се суочавају са изазовима приликом прегледа великог броја пројеката без алата показују већу спремност за коришћење алата (X11). Стога, коришћење алата за аутоматско тестирање може убрзати овај процес и обезбедити конзистентност у оцењивању. Са друге стране, овај корак може бити спроведен на традиционалан начин, краћим мануелним прегледом функционалности софтверског решења уз пропратну одбрану, односно испитивање студента о решењу.

3. корак: Спровођење статичке анализе кôда за добијање извештаја о квалитету кôда

Након верификације функционалности, наставник спроводи детаљну статичку анализу кôда како би проценио квалитет програмирања. Ова анализа обухвата преглед структуре кôда, употребу програмских образаца, оптимизацију, читљивост и придржавање стандарда. Резултати истраживања показују да наставници који сматрају методе за проверу плагијата и квалитета кôда адекватним чешће користе алате за њихову примену (X13). Добијени извештај омогућава наставнику да идентификује снаге и слабости у софтверском пројекту.

4. корак: Генерисање оцене на основу анализе кôда и провере функционалности

Коначна оцена студента се формира интеграцијом резултата из претходних корака. Оцена обухвата:

- Квалитет кôда: Оцењује се на основу извештаја статичке анализе, укључујући читљивост, ефикасност и придржавање стандарда.
- Функционалност: Проверава се да ли пројекат испуњава све задате функционалне захтеве и спецификације.
- Академски интегритет: Уколико је пројекат прошао проверу плагијата без индикација неакадемског понашања.

Овај приступ осигурава да оцена рефлектује све аспекте студентског рада, подстичући студенте не само да испуне задатак, већ и да га реализују на квалитетан и професионалан начин.

Имликације за образовне институције и наставнике:

1. Обука и техничка подршка за наставнике: Да би се олакшало усвајање корака, образовне институције треба да обезбеде адекватну обуку и техничку подршку за коришћење алата за проверу плагијата и статичку анализу кôда.
2. Промоција свести о важности плагијата и квалитета кôда: Подизање свести о значају академског интегритета и квалитета кôда може подстаћи наставнике да интегришу ове праксе у своју наставу. Ово је подржано резултатима који показују да перцепција важности проблема плагијата утиче на спремност за коришћење алата (X10 и X12).
3. Интеграција алата као стандардне праксе: Увођење алата за проверу плагијата и статичку анализу кôда као стандардног дела процеса оцењивања може побољшати објективност и ефикасност. Резултати указују на то да наставници који сматрају ове методе адекватним чешће користе поменуте алате (X13).

4. Прилагођавање процеса обиму рада: За наставнике који се суочавају са великим бројем пројеката, алати могу значајно смањити оптерећење и побољшати квалитет оцењивања. Ово је важно јер је утврђено да изазовност прегледа без алата утиче на спремност за њихову употребу (X11).

Применом конкретних корака у процесу оцењивања, наставници могу обезбедити свеобухватну и објективну процену студентских софтверских пројеката. Овај систематичан приступ подстиче развој техничких и етичких вештина код студената, што је од суштинске важности за њихову будућу каријеру у области софтверског инжењерства. Образовне институције играју кључну улогу у подршци овој иницијативи кроз обезбеђивање ресурса, обуке и промоције значаја академског интегритета.

5.3 Ограничење примене истраживачких модела

У поглављу 1.1. представљена је класификација софтверских пројеката према типу. Представљени модели проверени су и намењени за примену над академским (студентским) софтверским пројектима.

Предложени модели, смернице за организацију и поставку софтверских пројеката и смернице за оцењивање софтверских пројеката могу бити примењене и над другим типовима пројекта уз одговарајуће адаптације.

Сличности између типова пројеката:

- Фокус на квалитет кôда: Без обзира на тип пројекта, квалитет кôда је критичан за успех и одрживост софтверског производа. Статичка анализа кôда пружа објективне метрике које помажу у процени и унапређењу кôда.
- Поштовање стандарда и спецификација: Сви типови пројеката имају одређене стандарде и спецификације које треба да испуне. Модел може да оцени усклађеност кôда са тим стандардима, што је релевантно како за академске, тако и за комерцијалне и друге пројекте.
- Безбедност и безбедносне праксе: Безбедност је важна компонента сваког софтверског пројекта. Статичка анализа кôда може да идентификује потенцијалне безбедносне рањивости и помаже у побољшању безбедносних пракси.
- Унапређење процеса развоја: Независно од контекста, систематско оцењивање кôда и функционалности омогућава континуирано унапређење процеса развоја и квалитета производа.
- Потреба за метрикама и анализом: Сви типови пројеката имају потребу за мерљивим показатељима који се могу користити за доношење информисаних одлука и праћење напретка.

Наведене сличности указују на то да предложени модели и смернице за организацију и оцењивање софтверских пројеката, уз одговарајуће модификације и адаптације могу бити примењени и на остале типове софтверских пројеката. Један од праваца за наставак истраживања у овој области је управо могућност примене и адаптације предложеног модела на друге типове софтверских пројеката.

На пример, у корпоративном окружењу, провера плагијаризма није примарни фокус, али је изузетно важно осигурати поштовање лиценци и ауторских права. Компаније морају да гарантују да се у њиховом кôду не користи неовлашћени отворени кôд или кôд трећих лица без одговарајућих дозвола. Стога, модул за проверу плагијаризма из предложеног модела могао би бити модификован да обухвати:

- Анализу лиценци: Аутоматска провера компатибилности лиценци коришћених библиотека и компоненти.
- Детекцију неовлашћеног кôда: Идентификација кôда који је потенцијално копиран из неауторизованих извора.

Модел показује добре резултате и широку могућност примене над академским (студентским) типом пројеката, међутим, могуће је осигурати његову применљивост и на осталим типовима софтверских пројеката. Адаптацијом модела и смерница омогућило би организацијама и појединцима да побољшају квалитет свог софтвера, поштовање законских оквира и ефикасност развојних процеса, без обзира на специфичности пројекта или окружења у ком се развија. Ипак, да би се овај модел успешно и ефективно применио у другим контекстима, неопходна су додатна истраживања.

6. Закључна разматрања и правци даљег истраживања

У првим поглављима ове дисертације, кроз уводна разматрања и преглед теоријских основа и стања у области, детаљно су обрађени основни појмови квалитета кôда, статичке анализе кôда и алата који се у ту сврху користе. Представљена литература обухвата могуће примене статичке анализе кôда, потенцијалне одреднице квалитета кôда и важност провере плагијаризма, као и потенцијал за аутоматско оцењивање софтверских пројеката. Идентификовани су постојећи проблеми у процесу прегледања софтверских пројеката и анализиране су могућности за њихово решавање. На основу предложених модела, спроведена су бројна истраживања чији су резултати представљени и дискутовани.

Кључни резултати остварени у оквиру ове дисертације су следећи:

1. Идентификација одредница квалитета кôда и употребе статичке анализе кôда: Прегледом релевантне литературе уочене су кључне одреднице које утичу на квалитет кôда, као и потенцијали примене статичке анализе кôда у процени тог квалитета. Такође су идентификовани изазови у вези са плагијаризмом и прегледом великог броја софтверских пројеката.
2. Обухватање свих варијабли квалитета кôда: За разлику од претходних истраживања која су се фокусирала на појединачне аспекте квалитета кôда, ова дисертација је обухватила све варијабле идентификоване у прегледу стања у области као потенцијалне одреднице квалитета кôда, пружајући тако свеобухватнији приступ овој теми.
3. Предлог модела оцењивања софтверских пројеката: Развијен је модел који обезбеђује објективност, тачност и академски интегритет у процесу оцењивања студентских софтверских пројеката. Овај модел интегрише употребу алата за статичку анализу кôда и проверу плагијаризма, чиме се унапређује квалитет и ефикасност оцењивања.
4. Примена модела на различите типове софтверских пројеката: Иако је модел првенствено осмишљен за академско окружење, показано је да се може успешно применити и на друге типове софтверских пројеката, проширујући тако његову корисност и применљивост у ширем контексту софтверског инжењерства.
5. Смернице за организацију и поставку софтверских пројеката: Издвојене су и детаљно описане смернице и кораци за организацију и структуру софтверских пројеката. Ове смернице осигуравају да су идентификоване одреднице квалитета кôда имплементиране на начин који омогућава развој софтверских решења највишег квалитета.
6. Смернице за спровођење поступка оцењивања: Формулисане су јасне инструкције за спровођење процеса оцењивања софтверских пројеката према предложеном моделу. Ове смернице омогућавају објективно и прецизно оцењивање, укључујући и проверу плагијаризма, чиме се додатно јача академски интегритет и квалитет образовног процеса.

Ови резултати представљају значајан допринос области оцењивања софтверских пројеката, нудећи свеобухватан приступ који комбинује теоријске основе са практичним смерницама за имплементацију. Предложени модел и смернице могу служити као основа за будућа истраживања и унапређења у овој области, са потенцијалом да побољшају квалитет образовања и професионалне праксе у софтверском инжењерству.

6.1 Правци даљег истраживања

Будућа истраживања би могла да се одвијају у четири кључна правца, и то:

- Развој прототипа који би подржао предложени модел оцењивања пројеката.
- Укључивање већег броја институција и њихових пројеката ради додатног испитивања одредница квалитета и проблема плагијаризма.
- Поређење добијених резултата са реалним пројектима из индустрије.
- Додатно прилагођавање предложеног модела на академским примерима за друге типове софтверских пројеката.

Први правац подразумева развој специјализованог софтверског алата који би имплементирао предложени модел оцењивања студентских софтверских пројеката. Такав алат би интегрисао функционалности за проверу плагијаризма у кôду, спровођење статичке анализе кôда и аутоматско генерисање оцена на основу дефинисаних критеријума квалитета. Развој овог алата би омогућио стандардизацију и аутоматизацију процеса оцењивања, чиме би се повећала ефикасност и објективност наставника. Поред тога, алат би могао бити дизајниран да подржава различите програмске језике и платформе, што би га учинило применљивим у ширем академском и индустријском контексту.

Други правац се односи на укључивање већег броја образовних институција и анализу њихових студентских софтверских пројеката. Ово би омогућило проширење узорка и повећање валидности и поузданости добијених резултата. Сарадња са различитим институцијама би допринела бољем разумевању начин на који се одреднице квалитета кôда и проблем плагијаризма манифестују у различитим образовним окружењима и курикулумима. Поређење резултата између институција би могло открити заједничке изазове и специфичности, што би допринело креирању универзалних смерница за побољшање наставне праксе и политика академског интегритета.

Још један од могућих праваца подразумева поређење одредница квалитета кôда и степена плагијаризма у студентским пројектима са реалним софтверским пројектима из индустрије. Овај приступ би омогућио да се утврди у којој мери академски стандарди и праксе одговарају стварним потребама и очекивањима софтверске индустрије. Резултати таквог поређења могли би да пруже вредне увиде за унапређење курикулума и наставних метода, како би студенти били боље припремљени за професионалне изазове. Такође, идентификовање сличности и разлика у квалитету кôда би могло допринети развоју програма обуке и професионалног усавршавања.

Последњи правац даљег истраживања укључује додатно прилагођавање предложеног модела на академским примерима за друге типове софтверских пројеката. Прилагођавање модела овим специфичностима може захтевати редефинисање критеријума квалитета кôда и избор одговарајућих алата за статичку анализу. Даље истраживање у овом правцу би омогућило да се модел учини универзалнијим и применљивијим у разним образовним и професионалним сценаријима, чиме би се повећала његова вредност и релевантност.

Закључно, описани правци даљег истраживања пружају широку платформу за продубљивање разумевања и унапређење процеса оцењивања софтверских пројеката. Развој специјализованог алата би омогућио практичну примену предложеног модела и повећао ефикасност рада наставника. Укључивање већег броја институција и поређење са индустријским стандардима би допринело валидности и релевантности резултата, као и побољшању образовних пракси. Прилагођавање модела различитим типовима пројеката би осигурало његову флексибилност и применљивост у различитим контекстима. Све ово заједно може допринети стварању стандардизованог, објективног и ефикасног система оцењивања који промовише квалитет кôда и одржава академски интегритет.

7. Литература

[1] B. Chess and J. West, *Secure Programming with Static Code Analysis*. Addison-Wesley Professional, 2007.

[2] S. Hamer, C. Quesada-López, A. Martinez, and M. Jenkins, "Using git metrics to measure students' and teams' code contributions in software development projects," *CLEI Electron. J.*, vol. 24, Jul. 2021, doi: 10.19153/cleiej.24.2.8.

[3] J. Börstler *et al.*, "Developers talking about code quality," *Empir. Softw. Eng.*, vol. 28, no. 6, p. 128, Sep. 2023, doi: 10.1007/s10664-023-10381-0.

[4] A. Name, "Software Quality: The Complexity and Challenges of Software Engineering and Software Quality in the Cloud," in *Proceedings of the International Conference on Software Quality Days (SWQD)*, Springer, 2019, pp. 45–58. doi: 10.1007/978-3-030-13229-6_4.

[5] A. Møller and M. I. Schwartzbach, *Static Program Analysis*. Department of Computer Science, Aarhus University, 2020. [Online]. Available: https://cs.au.dk/~amoeller/spa/

[6] T. Delev and D. Gjorgjevikj, "Static analysis of source code written by novice programmers," in *2017 IEEE Global Engineering Education Conference (EDUCON)*, 2017, pp. 825–830. doi: 10.1109/EDUCON.2017.7942942.

[7] M. Ichinco, A. Zemach, and C. Kelleher, "Towards generalizing expert programmers' suggestions for novice programmers," in *2013 IEEE Symposium on Visual Languages and Human Centric Computing*, 2013, pp. 143–150. doi: 10.1109/VLHCC.2013.6645259.

[8] O. M. Mirza, M. Joy, and G. Cosma, "Style Analysis for Source Code Plagiarism Detection — An Analysis of a Dataset of Student Coursework," in *2017 IEEE 17th International Conference on Advanced Learning Technologies (ICALT)*, 2017, pp. 296–297. doi: 10.1109/ICALT.2017.117.

[9] M. Novak, M. S. Joy, and O. M. Mirza, "Improved plagiarism detection with collaboration network visualization based on source-code similarity," in *2021 IEEE Technology & Engineering Management Conference - Europe (TEMSCON-EUR)*, 2021, pp. 1–6. doi: 10.1109/TEMSCON-EUR52034.2021.9488644.

[10] I. Mekterović, L. Brkić, B. Milašinović, and M. Baranović, "Building a Comprehensive Automated Programming Assessment System," *IEEE Access*, vol. 8, pp. 81154–81172, 2020, doi: 10.1109/ACCESS.2020.2990980.

[11] I. Mekterović, L. Brkić, M. Fertalj, and M. Fabijanić, "Interactive Programming Tutorials in Automated Programming Assessment System Edgar," in *2024 47th MIPRO ICT and Electronics Convention (MIPRO)*, May 2024, pp. 218–223. doi: 10.1109/MIPRO60963.2024.10569406.

[12] D. Stefanović, D. Nikolić, D. Dakic, I. Spasojević, and S. Ristic, "Static Code Analysis Tools: A Systematic Literature Review," 2020, pp. 0565–0573. doi: 10.2507/31st.daaam.proceedings.078.

[13] D. Nikolić, D. Stefanović, D. Dakić, S. Sladojević, and S. Ristić, "Analysis of the Tools for Static Code Analysis," in *2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH)*, 2021, pp. 1–6. doi: 10.1109/INFOTEH51037.2021.9400688.

[14] D. Stefanović, D. Nikolić, S. Havzi, T. Vučković, and D. Dakic, "Identification of strategies over tools for static code analysis," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 1163, p. 012012, Aug. 2021, doi: 10.1088/1757-899X/1163/1/012012.

[15] D. Nikolić, S. Havzi, D. Dakić, T. Lolić, and D. Stefanović, “EVALUATION OF STRATEGIES OVER STATIC CODE ANALYSIS TOOLS,” 2021.

[16] A. Komosar, S. Kijanovic, V. Mandic, D. Nikolic, and T. Vuckovic, “On the Application of Static Code Analysis Tools in the Serbian IT Industry: An Empirical Study,” in *17th IADIS International Conference Information Systems 2024*, Faculty of Technical Sciences, University of Novi Sad, Trg Dositeja Obradovica 6, Novi Sad, Serbia, 2024, pp. 76–83.

[17] D. Nikolić, D. Stefanović, M. Nikolić, D. Dakić, M. Stefanović, and S. Koprivica, “Uncovering Determinants of Code Quality In Education Via Static Code Analysis,” *IEEE Access*, 2024.

[18] Y. Bai, T. Wang, and H. Wang, “Amelioration of Teaching Strategies by Exploring Code Quality and Submission Behavior,” *IEEE Access*, vol. 7, pp. 152744–152754, 2019, doi: 10.1109/ACCESS.2019.2948640.

[19] M. Beller, R. Bholanath, S. Mcintosh, and A. Zaidman, *Analyzing the State of Static Analysis: A Large-Scale Evaluation in Open Source Software*. 2016. doi: 10.1109/SANER.2016.105.

[20] L. Gren and V. Antinyan, “On the relation between unit testing and code quality,” in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, IEEE, 2017, pp. 52–56.

[21] JUSST, “A Review of Static Code Analysis Methods for Detecting Security Flaws.” 2021. [Online]. Available: https://jusst.org/wp-content/uploads/2021/06/A-Review-of-Static-Code-Analysis-Methods-for-Detecting-Security-Flaws.pdf

[22] In-Com, “Mastering Static Source Code Analysis for Secure Coding.” 2023. [Online]. Available: https://www.in-com.com/static-source-code-analysis-for-secure-coding

[23] P. Louridas, “Static code analysis,” *IEEE Softw.*, vol. 23, no. 4, pp. 58–61, 2006, doi: 10.1109/MS.2006.101.

[24] Codase, “What is Static Code Analysis?” 2023. [Online]. Available: https://www.codase.com/static-code-analysis

[25] D. A. Kumar, “Vulnerability detection in software applications using static code analysis,” *J. Theor. Appl. Inf. Technol.*, vol. 102, no. 4, 2024, [Online]. Available: https://jatit.info/index.php/jatit/article/view/3

[26] S. Heckman and L. Williams, “A systematic literature review of actionable alert identification techniques for automated static code analysis,” *Inf. Softw. Technol.*, vol. 53, no. 4, pp. 363–387, 2011.

[27] A. Băicoianu and I. Plajer, “Considerations on efficient lexical analysis in the context of compiler design,” *Bull. Transilv. Univ. Brasov Ser. III Math. Comput. Sci.*, pp. 159–168, Dec. 2023, doi: 10.31926/but.mif.2023.3.65.2.14.

[28] W. Ma *et al.*, “LMs: Understanding Code Syntax and Semantics for Code Analysis.” 2024. [Online]. Available: https://arxiv.org/abs/2305.12138

[29] G. Fan, X. Xie, X. Zheng, Y. Liang, and P. Di, “Static Code Analysis in the AI Era: An In-depth Exploration of the Concept, Function, and Potential of Intelligent Code Analysis Agents.” 2023. [Online]. Available: https://arxiv.org/abs/2310.08837

[30] SonarSource, “SonarQube.” [Online]. Available: https://www.sonarsource.com

[31] PMD Project, “PMD.” [Online]. Available: https://pmd.github.io

[32] K. F. Tómasdóttir, M. Aniche, and A. Van Deursen, “The adoption of javascript linters in practice: A case study on eslint,” *IEEE Trans. Softw. Eng.*, vol. 46, no. 8, pp. 863–891, 2018.

[33] K. F. Tómasdóttir, M. Aniche, and A. Van Deursen, “Why and how JavaScript developers use linters,” in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, 2017, pp. 578–589.

[34] The FindBugs Project, “FindBugs.” [Online]. Available: http://findbugs.sourceforge.net

[35] “ISO/IEC 25010:2023, Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.” International Organization for Standardization, 2023. [Online]. Available: https://www.iso.org/standard/35733.html

[36] F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, Anniversary Edition. Reading, Massachusetts: Addison-Wesley Publishing Company, 1995.

[37] I. Sommerville, *Software Engineering*, 10th ed. Harlow, United Kingdom: Pearson Education Limited, 2016.

[38] B. Kitchenham and S. Charters, “Guidelines for performing Systematic Literature Reviews in Software Engineering,” vol. 2, Jan. 2007.

[39] A. Mandal, D. Mohan, R. Jetley, S. Nair, and M. D’Souza, “A Generic Static Analysis Framework for Domain-specific Languages,” in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2018, pp. 27–34. doi: 10.1109/ETFA.2018.8502576.

[40] M. A. K. Maduranga, D. C. Mahagamage, P. I. Madhavi, J. A. H. Madushan, and C. Wijesiriwardana, “DOMAIN SPECIFIC INFRASTRUCTURE FOR CODE SMELL DETECTION IN LARGE-SCALE SOFTWARE SYSTEMS,” 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:20983165

[41] I. Ruiz-Rube, T. Person, J. M. Dodero, J. M. Mota, and J. M. Sánchez-Jara, “Applying static code analysis for domain-specific languages,” *Softw. Syst. Model.*, vol. 19, no. 1, pp. 95–110, Jan. 2020, doi: 10.1007/s10270-019-00729-w.

[42] A. Kaur and R. Nayyar, “A Comparative Study of Static Code Analysis tools for Vulnerability Detection in C/C++ and JAVA Source Code,” *Third Int. Conf. Comput. Netw. Commun. CoCoNet19*, vol. 171, pp. 2023–2029, Jan. 2020, doi: 10.1016/j.procs.2020.04.217.

[43] D. Marcilio, C. A. Furia, R. Bonifácio, and G. Pinto, “SpongeBugs: Automatically generating fix suggestions in response to static code analysis warnings,” *J. Syst. Softw.*, vol. 168, p. 110671, Oct. 2020, doi: 10.1016/j.jss.2020.110671.

[44] E. Aivaloglou and A. van der Meulen, “An Empirical Study of Students’ Perceptions on the Setup and Grading of Group Programming Assignments,” *ACM Trans Comput Educ*, vol. 21, no. 3, Mar. 2021, doi: 10.1145/3440994.

[45] N. Meghanathan, “Identification and Removal of Software Security Vulnerabilities using Source Code Analysis: A Case Study on a Java File Writer Program with Password Validation Features,” *J. Softw.*, vol. 8, Oct. 2013, doi: 10.4304/jsw.8.10.2412-2424.

[46] K. Goseva-Popstojanova and A. Perhinschi, “On the capability of static code analysis to detect security vulnerabilities,” *Inf. Softw. Technol.*, vol. 68, pp. 18–33, Dec. 2015, doi: 10.1016/j.infsof.2015.08.002.

[47] C. Chahar, V. Singh, and M. Das, “Code Analysis for Software and System Security Using Open Source Tools,” *Inf. Secur. J. Glob. Perspect.*, vol. 21, Jan. 2012, doi: 10.1080/19393555.2012.727132.

[48] P. Seshagiri, A. Vazhayil, and P. Sriram, “AMA: Static Code Analysis of Web Page for the Detection of Malicious Scripts,” *Procedia Comput. Sci.*, vol. 93, pp. 768–773, Dec. 2016, doi: 10.1016/j.procs.2016.07.291.

[49] T. Vert, T. Krikun, and M. Glukhikh, “Detection of incorrect pointer dereferences for C/C++ programs using static code analysis and logical inference,” in *2013 Tools & Methods of Program Analysis*, 2013, pp. 78–82. doi: 10.1109/TMPA.2013.7163724.

[50] L. Xin and C. Wandong, “A program vulnerabilities detection frame by static code analysis and model checking,” in *2011 IEEE 3rd International Conference on Communication Software and Networks*, 2011, pp. 130–134. doi: 10.1109/ICCSN.2011.6013559.

[51] N. Antunes and M. Vieira, “Benchmarking Vulnerability Detection Tools for Web Services,” in *2010 IEEE International Conference on Web Services*, 2010, pp. 203–210. doi: 10.1109/ICWS.2010.76.

[52] F. Zampetti, S. Scalabrino, R. Oliveto, G. Canfora, and M. Di Penta, “How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines,” in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 334–344. doi: 10.1109/MSR.2017.2.

[53] Z. Zhioua and Y. Roudier, *Static Code Analysis for Software Security Verification: Problems and Approaches*. 2014. doi: 10.1109/COMPSACW.2014.22.

[54] A. Arusoaie, S. Ciobâca, V. Craciun, D. Gavrilut, and D. Lucanu, “A Comparison of Open-Source Static Analysis Tools for Vulnerability Detection in C/C++ Code,” in *2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 2017, pp. 161–168. doi: 10.1109/SYNASC.2017.00035.

[55] N. Manzoor, H. Munir, and M. Moayyed, “Comparison of Static Analysis Tools for Finding Concurrency Bugs,” in *2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops*, 2012, pp. 129–133. doi: 10.1109/ISSREW.2012.28.

[56] M. A. A. Mamun, A. Khanam, H. Grahn, and R. Feldt, “Comparing Four Static Analysis Tools for Java Concurrency Bugs,” 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:1684268

[57] G. Chatzieleftheriou and P. Katsaros, *Test-Driving Static Analysis Tools in Search of C Code Vulnerabilities*. 2011, p. 103. doi: 10.1109/COMPSACW.2011.26.

[58] J. García-Muñoz, M. García-Valls, and J. Escribano-Barreno, “Improved Metrics Handling in SonarQube for Software Quality Monitoring,” 2016, pp. 463–470. doi: 10.1007/978-3-319-40162-1_50.

[59] J. Xie, H. Lipford, and B.-T. Chu, “Evaluating interactive support for secure programming,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, in CHI ’12. New York, NY, USA: Association for Computing Machinery, 2012, pp. 2707–2716. doi: 10.1145/2207676.2208665.

[60] E. Penttilä, “Improving C++ Software Quality with Static Code Analysis,” 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:35583396

[61] K. Tsipenyuk, B. Chess, and G. McGraw, “Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors,” *IEEE Secur. Priv.*, vol. 3, no. 6, pp. 81–84, Nov. 2005.

[62] H. Prahofer, F. Angerer, R. Ramler, and F. Grillenberger, “Static Code Analysis of IEC 61131-3 Programs: Comprehensive Tool Support and Experiences from Large-Scale Industrial Application,” *IEEE Trans. Ind. Inform.*, vol. PP, pp. 1–1, Aug. 2016, doi: 10.1109/TII.2016.2604760.

[63] C. G. Von Wangenheim *et al.*, “CodeMaster–Automatic Assessment and Grading of App Inventor and Snap! Programs.,” *Inform. Educ.*, vol. 17, no. 1, pp. 117–150, 2018.

[64] B. Hass, C. Yuan, and Z. Li, *On the Automatic Assessment of Learning Outcome in Programming Techniques*. 2019, p. 278. doi: 10.1109/ISKE47853.2019.9170370.

[65] E. A. AlOmar, S. A. AlOmar, and M. W. Mkaouer, “On the use of static analysis to engage students with software quality improvement: An experience with PMD,” in *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, 2023, pp. 179–191. doi: 10.1109/ICSE-SEET58685.2023.00023.

[66] P. Ardimento, M. L. Bernardi, and M. Cimitile, “Software Analytics to Support Students in Object-Oriented Programming Tasks: An Empirical Study,” *IEEE Access*, vol. 8, pp. 132171–132187, 2020, doi: 10.1109/ACCESS.2020.3010172.

[67] M. Kern *et al.*, “Integrating Static Code Analysis Toolchains,” in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, 2019, pp. 523–528. doi: 10.1109/COMPSAC.2019.00080.

[68] R. Cardell-Oliver, “How can Software Metrics Help Novice Programmers?,” Jan. 2011, pp. 55–62.

[69] H.-M. Chen, B. Nguyen, and C.-R. Dow, “Code-quality evaluation scheme for assessment of student contributions to programming projects,” *J. Syst. Softw.*, vol. 188, p. 111273, Feb. 2022, doi: 10.1016/j.jss.2022.111273.

[70] S. A. Ebad, A. A. Darem, and J. H. Abawajy, “Measuring Software Obfuscation Quality–A Systematic Literature Review,” *IEEE Access*, vol. 9, pp. 99024–99038, 2021, doi: 10.1109/ACCESS.2021.3094517.

[71] B. Ray, D. Posnett, V. Filkov, and P. Devanbu, “A large scale study of programming languages and code quality in github,” *Proc FSE 2014*, pp. 155–165, Nov. 2014, doi: 10.1145/2635868.2635922.

[72] T. Freire and C. Rodríguez, “The transformation to an online course in higher education results in better student academic performance,” *RIED Rev. Iberoam. Educ. Distancia*, vol. 25, no. 1, pp. 299–322, 2022.

[73] O. S. Kaya and H. Bicen, “Study of augmented reality applications use in education and its effect on the academic performance,” *Int. J. Distance Educ. Technol. IJDET*, vol. 17, no. 3, pp. 25–36, 2019.

[74] B. Ali and Dr. F. Baig, “The Impact of Educational Videos on the Academic Performance of University Students in Distance Learning,” vol. 6, pp. 1233–1249, Dec. 2022.

[75] S. Bhatia and J. Malhotra, “A survey on impact of lines of code on software complexity,” in *2014 International Conference on Advances in Engineering & Technology Research (ICAETR - 2014)*, 2014, pp. 1–4. doi: 10.1109/ICAETR.2014.7012875.

[76] H. Hokka, F. Dobslaw, and J. Bengtsson, “Linking Developer Experience to Coding Style in Open-Source Repositories,” in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Mar. 2021, pp. 516–520. doi: 10.1109/SANER50967.2021.00057.

[77] Y. Cho, J. -H. Kwon, J. Yi, and I. -Y. Ko, “Extending Developer Experience Metrics for Better Effort-Aware Just-In-Time Defect Prediction,” *IEEE Access*, vol. 10, pp. 128218–128231, 2022, doi: 10.1109/ACCESS.2022.3227339.

[78] S. Sripada, R. Reddy, and A. Sureka, *In Support of Peer Code Review and Inspection in an Undergraduate Software Engineering Course*. 2015, p. 6. doi: 10.1109/CSEET.2015.8.

[79] J. Ludwig, S. Xu, and F. Webber, *Compiling static software metrics for reliability and maintainability from GitHub repositories*. 2017, p. 9. doi: 10.1109/SMC.2017.8122569.

[80] R. Harrison, L. G. Smaraweera, M. R. Dobie, and P. H. Lewis, "Comparing Programming Paradigms: an Evaluation of Functional and Object-Oriented Programs," *Softw. Eng. J.*, vol. 11, pp. 247–254, Aug. 1996, doi: 10.1049/sej.1996.0030.

[81] P. Bhattacharya and I. Neamtiu, "Assessing programming language impact on development and maintenance: a study on c and c++," in *2011 33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 171–180. doi: 10.1145/1985793.1985817.

[82] "Bitbucket." [Online]. Available: https://bitbucket.org

[83] "GitHub." [Online]. Available: https://github.com

[84] D. Kerschbaumer, "The Code to Success: Developing a Student Source Code Analysis Tool to Predict Course Performance and Identify Key Concepts in Programming Education," 2023. doi: 10.13140/RG.2.2.28072.75528.

[85] C. Alexandra-Cristina and A.-C. Olteanu, "Material Survey on Source Code Plagiarism Detection in Programming Courses," in *2022 International Conference on Advanced Learning Technologies (ICALT)*, 2022, pp. 387–389. doi: 10.1109/ICALT55010.2022.00119.

[86] Y. Amaliah, W. Musu, Suprianto, and M. Fadlan, "Auto Clustering Source Code To Detect Plagiarism Of Student Programming Assignments in Java Programming Language," in *2021 3rd International Conference on Cybernetics and Intelligent System (ICORIS)*, 2021, pp. 1–6. doi: 10.1109/ICORIS52787.2021.9649465.

[87] M. Joy, G. Cosma, J. Y.-K. Yau, and J. Sinclair, "Source Code Plagiarism—A Student Perspective," *IEEE Trans. Educ.*, vol. 54, no. 1, pp. 125–132, 2011, doi: 10.1109/TE.2010.2046664.

[88] H.-M. Chen, B.-A. Nguyen, Y.-X. Yan, and C.-R. Dow, "Analysis of Learning Behavior in an Automated Programming Assessment Environment: A Code Quality Perspective," *IEEE Access*, vol. 8, pp. 167341–167354, 2020, doi: 10.1109/ACCESS.2020.3024102.

[89] F. Engelberger, P. Galaz-Davison, G. Bravo, M. Rivera, and C. A. Ramírez-Sarmiento, "Developing and Implementing Cloud-Based Tutorials That Combine Bioinformatics Software, Interactive Coding, and Visualization Exercises for Distance Learning on Structural Bioinformatics," *J. Chem. Educ.*, vol. 98, no. 5, pp. 1801–1807, May 2021, doi: 10.1021/acs.jchemed.1c00022.

[90] L. Brkic, I. Mekterovic, M. Fertalj, and D. Mekterović, "Peer assessment methodology of open-ended assignments: Insights from a two-year case study within a university course using novel open source system," *Comput. Educ.*, vol. 213, p. 105001, Feb. 2024, doi: 10.1016/j.compedu.2024.105001.

[91] T. Staubitz, H. Klement, J. Renz, R. Teusner, and C. Meinel, "Towards practical programming exercises and automated assessment in Massive Open Online Courses," in *2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, Dec. 2015, pp. 23–30. doi: 10.1109/TALE.2015.7386010.

[92] JASP Team, "JASP (Version 0.19.0)[Computer software]." 2024. [Online]. Available: https://jasp-stats.org/

[93] "SonarQube Documentation." 2024. [Online]. Available: https://www.sonarqube.org/docs/

[94] C. -H. Cao, Y. -N. Tang, H. Zhou, Y. -L. Li, and Z. Marszałek, "DBSCAN-Based Automatic De-Duplication for Software Quality Inspection Data," *IEEE Access*, vol. 11, pp. 17882–17890, 2023, doi: 10.1109/ACCESS.2022.3164192.

[95] D. K. K. Shyamal, P. P. G. D. Asanka, and D. Wickramaarachchi, “A Comprehensive Approach to Evaluating Software Code Quality Through a Flexible Quality Model,” in *2023 International Research Conference on Smart Computing and Systems Engineering (SCSE)*, Jun. 2023, pp. 1–8. doi: 10.1109/SCSE59836.2023.10215004.

[96] M. A. A. Hilmi, A. Puspaningrum, Darsih, D. O. Siahaan, H. S. Samosir, and A. S. Rahma, “Research Trends, Detection Methods, Practices, and Challenges in Code Smell: SLR,” *IEEE Access*, vol. 11, pp. 129536–129551, 2023, doi: 10.1109/ACCESS.2023.3334258.

[97] S. Dewangan, R. S. Rao, A. Mishra, and M. Gupta, “A Novel Approach for Code Smell Detection: An Empirical Study,” *IEEE Access*, vol. 9, pp. 162869–162883, 2021, doi: 10.1109/ACCESS.2021.3133810.

[98] R. Kumar, *Research methodology : a step-by-step guide for beginners*, 3rd ed. Los Angeles: SAGE, 2011.

[99] B. A. Kitchenham, “Evaluating software engineering methods and tool part 1: The evaluation context and evaluation methods,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 21, no. 1, pp. 11–14, 1996.

[100] B. A. Kitchenham, “Evaluating software engineering methods and tools part 12: evaluating DESMET,” *SIGSOFT Softw Eng Notes*, vol. 23, no. 5, pp. 21–24, Sep. 1998, doi: 10.1145/290249.290255.

[101] Cppcheck Development Team, “CppCheck.” [Online]. Available: http://cppcheck.sourceforge.net

[102] J. I. E. Hoffman, “Biostatistics for Medical and Biomedical Practitioners,” pp. 1–744, Jan. 2015.

[103] A. T. Jebb, V. Ng, and L. Tay, “A Review of Key Likert Scale Development Advances: 1995–2019,” *Front. Psychol.*, vol. 12, 2021, doi: 10.3389/fpsyg.2021.637547.

[104] R. M. Groves, F. J. Fowler, M. P. Couper, J. M. Lepkowski, E. Singer, and R. Tourangeau, *Survey Methodology*. in Wiley Series in Survey Methodology. Wiley, 2009. [Online]. Available: https://books.google.rs/books?id=HXoSpXvo3s4C

[105] I. Etikan, “Comparison of Convenience Sampling and Purposive Sampling,” *Am. J. Theor. Appl. Stat.*, vol. 5, p. 1, Jan. 2016, doi: 10.11648/j.ajtas.20160501.11.

[106] S. Campbell *et al.*, “Purposive sampling: complex or simple? Research case examples,” *J. Res. Nurs.*, vol. 25, no. 8, pp. 652–661, Dec. 2020, doi: 10.1177/1744987120927206.

[107] J. Michell, “Measurement Scales and Statistics. A Clash of Paradigms,” *Psychol. Bull.*, vol. 100, pp. 398–407, Nov. 1986, doi: 10.1037/0033-2909.100.3.398.

Прилог А - Опсервације ИМ1

Успех студента		Студент	Квалитет кôда					Опис пројекта						
ОС	**БРП**	**Пол**	**Б**	**Д(%)**	**Р**	**CS**	**БП**	**УСАК**	**УАВ**	**АГ**	**ВП**	**НОН**	**ТС**	**ИТ**
10	2	Ж	201	11	0	1900	6	нк.	кор.	III	32000	уживо	тимски	*.net*
10	2	Ж	201	11	0	1900	6	нк.	кор.	III	32000	уживо	пој.	*.net*
10	2	Ж	201	11	0	1900	6	нк.	кор.	III	32000	уживо	тимски	*.net*
10	2	Ж	201	11	0	1900	6	нк.	кор.	III	32000	уживо	тимски	*.net*
8	2	Ж	6	18	0	545	12	нк.	кор.	III	23000	уживо	тимски	*.net*
8	2	М	6	18	0	545	12	нк.	кор.	III	23000	уживо	тимски	*.net*
8	2	М	6	18	0	545	12	нк.	кор.	III	23000	уживо	тимски	*.net*
7	2	М	6	18	0	545	12	нк.	кор.	III	23000	уживо	тимски	*.net*
10	1	М	39	0.8	0	1100	17	нк.	кор.	III	19000	уживо	тимски	*.net*
10	1	М	39	0.8	0	1100	17	нк.	кор.	III	19000	уживо	тимски	*.net*
10	1	Ж	61	4.5	0	293	8	нк.	кор.	IV	22000	уживо	тимски	*angular*
10	1	М	61	4.5	0	293	8	нк.	кор.	IV	22000	уживо	тимски	*angular*

8	1	М	61	4.5	0	293	8	нк.	кор.	IV	22000	уживо	тимски	*angular*
8	1	Ж	61	4.5	0	293	8	нк.	кор.	IV	22000	уживо	тимски	*angular*
9	1	М	61	4.5	0	293	8	нк.	кор.	IV	22000	уживо	тимски	*angular*
10	1	М	61	4.5	0	293	8	нк.	кор.	IV	22000	уживо	тимски	*angular*
10	1	М	61	4.5	0	293	8	нк.	кор.	IV	22000	уживо	тимски	*angular*
10	1	М	61	4.5	0	293	8	нк.	кор.	IV	22000	уживо	тимски	*angular*
9	1	М	61	4.5	0	293	8	нк.	кор.	IV	22000	уживо	тимски	*angular*
9	1	М	61	4.5	0	293	8	нк.	кор.	IV	22000	уживо	тимски	*angular*
9	1	Ж	61	4.5	0	293	8	нк.	кор.	IV	22000	уживо	тимски	*angular*
7	1	Ж	61	4.5	0	293	8	нк.	кор.	IV	22000	уживо	тимски	*angular*
9	1	Ж	61	4.5	0	293	8	нк.	кор.	IV	22000	уживо	тимски	*angular*
9	1	М	61	4.5	0	293	8	нк.	кор.	IV	22000	уживо	тимски	*angular*
10	1	М	61	4.5	0	293	8	нк.	кор.	IV	22000	уживо	тимски	*angular*
10	1	Ж	28	20.4	0	742	92	нк.	кор.	III	6400	уживо	пој.	*java*
9	3	М	28	25.2	0	1200	22	нк.	кор.	III	7800	уживо	пој.	*java*

7	12	М	7	45.9	0	2600	0	нк.	кор.	III	24000	уживо	пој.	*java*
7	7	М	15	14	0	153	19	нк.	кор.	III	3500	уживо	пој.	*java*
10	1	Ж	21	1.2	0	135	4	нк.	кор.	III	3100	уживо	пој.	*java*
7	4	Ж	28	13.4	0	241	7	нк.	кор.	III	3900	уживо	пој.	*java*
9	3	Ж	20	13.4	0	198	7	нк.	кор.	III	3900	уживо	пој.	*java*
10	2	Ж	28	12	0	252	7	нк.	кор.	III	4000	уживо	пој.	*java*
7	4	М	15	8.4	0	276	12	нк.	кор.	III	4400	уживо	пој.	*java*
8	2	М	20	11.9	0	406	37	нк.	кор.	III	5000	уживо	пој.	*java*
6	4	Ж	30	10.6	0	289	12	нк.	кор.	III	4700	уживо	пој.	*java*
9	2	Ж	24	10.1	0	386	17	нк.	кор.	III	4800	уживо	пој.	*java*
НП	НП	М	17	22.1	0	406	8	нк.	кор.	III	4100	уживо	пој.	*java*
10	1	М	21	14.0	0	321	13	нк.	кор.	III	3800	онлајн	пој.	*java*
10	1	М	13	14.3	0	153	0	нк.	кор.	III	4700	онлајн	пој.	*java*
9	2	М	8	0	0	2000	0	кор.	кор.	IV	12000	уживо	тимски	*.net*
9	2	Ж	8	0	0	2000	0	кор.	кор.	IV	12000	уживо	тимски	*.net*

8	2	Ж	8	0	0	2000	0	кор.	кор.	IV	12000	уживо	тимски	*.net*
9	2	Ж	8	0	0	2000	0	кор.	кор.	IV	12000	уживо	тимски	*.net*
9	2	Ж	8	0	0	2000	0	кор.	кор.	IV	12000	уживо	тимски	*.net*
8	2	М	8	0	0	2000	0	кор.	кор.	IV	12000	уживо	тимски	*.net*
НП	2	М	8	0	0	2000	0	кор.	кор.	IV	12000	уживо	тимски	*.net*
8	2	М	8	0	0	2000	0	кор.	кор.	IV	12000	уживо	тимски	*.net*
9	1	Ж	0	10.5	0	0	0	кор.	кор.	IV	3500	уживо	тимски	*.net*
10	1	Ж	0	10.5	0	0	0	кор.	кор.	IV	3500	уживо	тимски	*.net*
9	1	М	0	10.5	0	0	0	кор.	кор.	IV	3500	уживо	тимски	*.net*
7	1	М	0	10.5	0	0	0	кор.	кор.	IV	3500	уживо	тимски	*.net*
9	1	Ж	0	10.5	0	0	0	кор.	кор.	IV	3500	уживо	тимски	*.net*
8	1	Ж	0	10.5	0	0	0	кор.	кор.	IV	3500	уживо	тимски	*.net*
10	1	М	0	5	10	3	5	нк.	кор.	III	440	онлајн	пој.	*spring, angular*
9	1	М	0	0	8	28	5	нк.	кор.	III	931	онлајн	пој.	*spring, angular*
8	3	М	1	0	8	56	7	нк.	кор.	III	663	онлајн	пој.	*spring, angular*

9	2	М	0	4	8	40	8	нк.	кор.	III	723	онлајн	пој.	*spring, angular*
8	4	Ж	0	10.3	7	39	4	нк.	кор.	III	720	онлајн	пој.	*spring, angular*
8	6	М	1	4.8	8	46	9	нк.	кор.	III	768	онлајн	пој.	*spring, angular*
НП	НП	Ж	0	0	0	1	0	нк.	кор.	III	64	онлајн	пој.	*spring, angular*
9	1	М	0	0	0	41	5	нк.	кор.	III	687	онлајн	пој.	*spring, angular*
8	6	М	0	15	8	37	5	нк.	кор.	III	639	онлајн	пој.	*spring, angular*
6	6	Ж	0	6	8	41	1	нк.	кор.	III	588	онлајн	пој.	*spring, angular*
8	4	М	0	0	8	48	8	нк.	кор.	III	746	онлајн	пој.	*spring, angular*
9	1	М	0	0	8	37	5	нк.	кор.	III	662	онлајн	пој.	*spring, angular*
10	1	Ж	0	0	8	42	9	нк.	кор.	III	706	онлајн	пој.	*spring, angular*
7	5	Ж	0	0	8	47	9	нк.	кор.	III	723	онлајн	пој.	*spring, angular*
10	1	М	14	0.5	0	36	0	нк.	кор.	III	1576	онлајн	пој.	*spring, angular*
НП	НП	Ж	0	18.5	0	34	0	нк.	нк.	II	1100	уживо	пој.	*wpf*
8	1	Ж	0	10.8	0	25	2	нк.	нк.	II	1300	уживо	пој.	*wpf*
10	1	Ж	0	4.5	0	55	20	нк.	нк.	II	1200	уживо	пој.	*wpf*

9	1	Ж	0	22.2	0	43	2	нк.	нк.	II	1200	уживо	пој.	*wpf*
9	1	М	0	10.7	0	20	1	нк.	нк.	II	1300	уживо	пој.	*wpf*
9	1	Ж	0	20.8	0	17	2	нк.	нк.	II	1300	уживо	пој.	*wpf*
НП	1	Ж	0	23.2	0	50	2	нк.	нк.	II	1900	уживо	пој.	*wpf*
10	1	Ж	0	22.8	0	27	1	нк.	нк.	II	1300	уживо	пој.	*wpf*
9	1	М	0	18.6	0	22	2	нк.	нк.	II	1300	уживо	пој.	*wpf*
9	1	М	0	31.8	0	40	2	нк.	нк.	II	1200	уживо	пој.	*wpf*
10	1	М	0	19.3	0	19	2	нк.	нк.	II	2000	уживо	пој.	*wpf*
10	1	М	0	15.4	0	20	2	нк.	нк.	II	1400	уживо	пој.	*wpf*
10	1	М	0	23.1	0	35	2	нк.	нк.	II	1300	уживо	пој.	*wpf*
10	1	Ж	0	22.9	0	44	2	нк.	нк.	II	1400	уживо	пој.	*wpf*
9	2	Ж	0	17.3	0	14	2	нк.	нк.	II	1400	уживо	пој.	*wpf*
10	1	Ж	9	0	0	21	0	нк.	нк.	I	723	уживо	пој.	*html/css*
8	2	М	5	4.9	0	40	1	нк.	нк.	I	710	уживо	пој.	*html/css*
10	1	Ж	16	2	0	24	0	нк.	нк.	I	1800	уживо	пој.	*html/css*

10	2	Ж	5	0	0	29	0	нк.	нк.	I	1400	уживо	пој.	*html/css*
10	1	Ж	12	0	0	11	1	нк.	нк.	I	1200	уживо	тимски	*html/css*
10	2	Ж	12	0	0	11	1	нк.	нк.	I	1200	уживо	тимски	*html/css*
9	2	М	13	2.1	0	22	0	нк.	нк.	I	1100	уживо	тимски	*html/css*
10	1	Ж	3	0	0	53	4	нк.	нк.	I	1100	уживо	тимски	*html/css*
10	1	Ж	3	0	0	53	4	нк.	нк.	I	1100	уживо	тимски	*html/css*
9	2	Ж	4	0	0	26	0	нк.	нк.	I	787	уживо	тимски	*html/css*
9	2	Ж	7	0	0	18	3	нк.	нк.	I	973	уживо	тимски	*html/css*
9	2	Ж	7	0	0	18	3	нк.	нк.	I	973	уживо	тимски	*html/css*
9	2	Ж	16	2	0	24	0	нк.	нк.	I	1800	уживо	тимски	*html/css*
10	2	Ж	9	0	0	17	1	нк.	нк.	I	916	уживо	тимски	*html/css*
9	1	Ж	9	0	0	17	1	нк.	нк.	I	916	уживо	тимски	*html/css*

*пој. – појединачно

кор. – коришћено

нк. – није коришћено

Прилог Б – Упитник ИМ2

Статичка анализа кôда у процесу оцењивања студентских софтверских пројеката

1. Увод

Драги учесници,

Хвала Вам на издвојеном времену за попуњавање упитника. Циљ овог истраживања је разумевање како се детектује плагијаризам у изворном кôду софтверских пројеката и у којој мери се проверава квалитет кôда у високошколском образовању. Ваши одговори биће драгоцени за наше истраживање. Упитник је намењен асистентима и професорима на Универзитетима у области софтверског инжењерства и биће Вам потребно 2-4 минута да га попуните.

Ваши одговори су анонимни и биће коришћени у академске сврхе.

Контакт: *nikolic.danilo@uns.ac.rs*

2. Општи подаци

1) Која је ваша универзитетска афилијација?

2) Да ли учествујете у одржавању наставе у области софтверског инжењерства?
- да и
- не.

3) У којој мери сте упознати са статичком анализом кôда и употребом алата за статичку анализу кôда?
- слабо,
- умерено добро,
- добро,
- веома добро и
- одлично.

3. Статичка анализа кôда и плагијаризам

1) Да ли вршите преглед софтверских пројеката употребом статичке анализе кôда?
- да и
- не.

2) Које алате користите за проверу квалитета кôда?
- *SonarQube,*
- *PMD,*
- *CheckStyle,*
- *Coverity,*
- *ESlint,*
- *FindBugs,*
- не користим,
- наведите други алат:

3) Да ли проверавате плагијаризам у кôду софтверских пројеката?
- да и
- не.

4) Уколико вршите проверу плагијаризма, које алате користите?
- *Turnitin,*
- *MOSS,*
- *Jplags,*
- не користим,
- наведите други алат:

5) Колико сматрате важним проблем плагијаризма у софтверским пројектима?
- није важно,
- мало је важно,
- делимично важно,
- важно и
- врло важно.

4. Аутоматско оцењивање и општи утисак

1) У којој мери вам је изазовно прегледање софтверских пројеката без употребе алата за проверу плагијаризма и аутоматско оцењивање софтверских пројеката?
- није изазовно,
- мало изазовно,
- делимично изазовно,
- изазовно и
- врло изазовно.

2) Оцените своју спремност да користите алат који би вам омогућио проверу плагијаризма и аутоматско оцењивање софтверских пројеката?
- не бих користио,
- користио бих у мањој мери,
- делимично бих користио,
- користио бих и
- користио бих у већој мери.

3) У којој мери сматрате релеватним и значајним прегледање софтверских пројеката употребом алата за проверу плагијаризма и аутоматско оцењивање?
- незначајно и нерелевантно,
- мало значајно и релевантно,
- делимично значајно и релевантно,
- значајно и релевантно и
- врло значајно и релевантно.

Прилог B - Статистичка анализа (ИМ1)

```
# importing libraries

require("ggplot2")
require("tidyr")
require("dplyr")
require("MASS")
require("blorr")
require("ModelMetrics")
require("pdp")
require("iml")
require("energy")

library("ggplot2")
library("tidyr")
library("dplyr")
library("MASS")
library("blorr")
library("ModelMetrics")
library("pdp")
library("iml")
library("scales")
library("energy")
library("MASS")
library("patchwork")
library("dplyr")

# loading data

path <- file.choose()
frame <- read.csv(path)

# numercial variables set

df_numerical<-pivot_longer(frame,cols=
c('logins','days_to_purchase','project_creation','add_client','time_entries_trac
ker','add_new_member','tracked_time_tracker'),
       names_to = "Variable", values_to = "Value")
```

```r
# categorical variables

df_categorical <- pivot_longer(frame, cols = c('continent', 'stripe_account'),
                               names_to = "Variable", values_to = "Value")

stats_categorical <- df_categorical %>%
  group_by(Variable, Value) %>%
  dplyr::summarize(
    Count = n()
  ) %>%
  ungroup()

df_cat_with_stats <- df_categorical %>%
  left_join(stats_categorical, by = c("Variable", "Value"))

categorical_plot <- ggplot(df_cat_with_stats, aes(x = Value)) +
  geom_bar(fill = "#4f81bd") +
  facet_wrap(~ Variable, scales = "free_x", strip.position = "bottom", nrow = 1,
ncol = 2) +
  theme_minimal() +
  labs(x = NULL, y = "count") +
  theme(
    text = element_text(family = "sans", size = 12),
    axis.text.x = element_text(angle = 45, hjust = 1, size = 10, face = "plain"),
    strip.placement = "outside",
    strip.background = element_blank(),
    strip.text = element_text(face = "plain", size = 12)
  )
categorical_plot
```

```r
# visualizing numerical variables distribution

stats_df <- df_numerical %>%
  group_by(Variable) %>%
  dplyr::summarize(
    Mean = mean(Value),
    Median = median(Value)
  ) %>%
  ungroup()
df_num_with_stats <- df_numerical %>%
  left_join(stats_df, by = "Variable")

numerical_plot <- ggplot(df_num_with_stats, aes(x = Value)) +
  geom_histogram(fill = "#4f81bd", bins = 8) +
   geom_vline(aes(xintercept = Mean, color = "Mean"), linetype = "dashed") +
   geom_vline(aes(xintercept = Median, color = "Median"), linetype = "solid") +
  scale_color_manual(values = c("Mean" = "black", "Median" = "black"),
                     labels = c(expression(mu), expression(tilde(x)))) +
   facet_wrap(~ Variable, scales = "free", strip.position = "bottom", nrow = 3,
 ncol = 3) +
  theme_minimal() +  labs(x = NULL, y = "count", color = "measures") +
          scale_x_continuous(labels = scales::comma) +
          theme(
            text = element_text(family = "sans", size = 12),
            axis.text.x = element_text(angle = 45, hjust = 1, size = 10, face =
 "plain"),
            strip.placement = "outside",
            strip.background = element_blank(),
            strip.text = element_text(face = "plain", size = 12)
          )

numerical_plot
```

```r
# distribution of the target classes across numerical variables

frame$target <- frame$target

df_numerical <- frame %>%
  pivot_longer(cols = c('logins', 'days_to_purchase', 'project_creation',
                        'add_client', 'time_entries_tracker', 'add_new_member',
                        'tracked_time_tracker'),
               names_to = "Variable", values_to = "Value")

numerical_target <- ggplot(df_numerical, aes(x = Value, y = as.factor(target),
color = as.factor(target))) +
  geom_jitter(width = 0.3, height = 0.1) +
  scale_color_manual(values = c("0" = "#4f81bd", "1" = "orange")) +
  facet_wrap(~ Variable, scales = "free_x", nrow = 3, ncol = 3, strip.position =
"bottom") +
  theme_minimal() +
  labs(x = "", y = expression(italic(y)), color = expression(italic(y))) +
  theme(
    text = element_text(family = "sans", size = 12),
    axis.text.x = element_text(angle = 45, hjust = 1, size = 10, face = "plain"),
    strip.background = element_blank(),
    strip.text = element_text(face = "plain", size = 12)
  ) +
  theme(
    strip.placement = "outside",
    strip.text.x = element_text(margin = margin(t = 10))
  )

numerical_target
# distribution of target classes across categorical variables' values

categorical_target <- ggplot(df_cat_with_stats, aes(x = Value, fill =
as.factor(target))) +
  geom_bar(position = "stack") + # 'stack' to place bars on top of each other for
different targets
  facet_wrap(~ Variable, scales = "free_x", strip.position = "bottom", nrow = 1,
ncol = 2) +
  theme_minimal() +
  labs(x = NULL, y = "count", fill = expression(italic(y))) +
  scale_fill_manual(values = c("0" = "#4f81bd", "1" = "orange")) +
  theme(
    text = element_text(family = "sans", size = 12),
    axis.text.x = element_text(angle = 45, hjust = 1, size = 10, face = "plain"),
    strip.placement = "outside",
    strip.background = element_blank(),
    strip.text = element_text(face = "plain", size = 12)
  )
categorical_target
```

```r
# dummies encoding the continent

dummies <- model.matrix(~ continent - 1, data = frame)

dummies_df <- as.data.frame(dummies)

colnames(dummies_df) <- gsub("continent", "", colnames(dummies_df))

frame_with_dummies <- cbind(frame, dummies_df)

# renaming variables to match the naming convention

frame_with_dummies <- frame_with_dummies %>%
  rename(private_email = emaail_encoded)

frame_with_dummies <- frame_with_dummies %>%
  rename(
    continent_frequency = continent_encoded,
    continent_africa = Africa,
    continent_asia = Asia,
    continent_australia = Australia,
    continent_europe = Europe,
    continent_north_america = `North America`,
    continent_south_america = `South America`
  )
```

```r
# visualizing distribution of encoded categorical variables

encoded_variables                                                    <-
frame_with_dummies[,c("continent_frequency","continent_africa", "continent_asia",
"continent_australia",                                    "continent_europe",
"continent_north_america","continent_south_america",
"private_email", "account_cake_ag")]

df_encoded <- encoded_variables %>%
  pivot_longer(cols = everything(), names_to = "Variable", values_to = "Value")

stats_encoded <- df_encoded %>%
  group_by(Variable) %>%
  dplyr::summarize(
    Mean = mean(Value),
    Median = median(Value)
  ) %>%
  ungroup()

df_encoded_with_stats <- df_encoded %>%
  left_join(stats_encoded, by = "Variable")

encoded_distribution <- ggplot(df_encoded_with_stats, aes(x = Value)) +
  geom_histogram(fill = "#4f81bd", bins = 8) +
  geom_vline(aes(xintercept = Mean, color = "Mean"), linetype = "dashed") +
  geom_vline(aes(xintercept = Median, color = "Median"), linetype = "solid") +
  scale_color_manual(values = c("Mean" = "black", "Median" = "black"),
                     labels = c(expression(mu), expression(tilde(x)))) +
  facet_wrap(~ Variable, scales = "free", strip.position = "bottom", nrow = 3, ncol
= 3) +
  theme_minimal() +
  labs(x = NULL, y = "count", color = "measures") +
  scale_x_continuous(labels = scales::comma) +
  theme(
    text = element_text(family = "sans", size = 12),
    axis.text.x = element_text(angle = 45, hjust = 1, size = 10, face = "plain"),
    strip.placement = "outside",
    strip.background = element_blank(),
    strip.text = element_text(face = "plain", size = 12)
  )

print(encoded_distribution
```

```r
# examining correlations

if (!requireNamespace("energy", quietly = TRUE)) {
  install.packages("energy")
}
library(energy)

compute_permutation_p_value <- function(x, y, num_permutations = 1000) {
  observed_dcor <- dcor(x, y)
  permuted_dcors <- replicate(num_permutations, {
  permuted_y <- sample(y)
  dcor(x, permuted_y)
  })
  p_value <- mean(permuted_dcors >= observed_dcor)
  return(p_value)
}

target <- frame_with_dummies$target

input_vars  <-  setdiff(names(frame_with_dummies),  c("target",  "stripe_account",
"continent"))
numeric_vars <- sapply(frame_with_dummies[input_vars], is.numeric)
numeric_input_vars <- input_vars[numeric_vars]

calc_dcor_and_pvalue <- function(var) {
  current_var <- frame_with_dummies[[var]]

  if (any(is.na(current_var)) || length(unique(current_var)) <= 1) {
    return(data.frame(variable = var, distance_correlation = NA, p_value = NA))
  }

  dcor_value <- dcor(current_var, target)
  p_value <- compute_permutation_p_value(current_var, target)

  return(data.frame(variable = var, distance_correlation = dcor_value, p_value =
p_value))
}

results <- do.call(rbind, lapply(numeric_input_vars, calc_dcor_and_pvalue))

results
```

```r
# examining linear relationship between X and y

independent_vars     <-     setdiff(names(frame_with_dummies),     c("workspace_id",
"country", "continent", "stripe_account", "target"))

df_long <- frame_with_dummies %>%
  dplyr::select(all_of(independent_vars), target) %>%
  pivot_longer(cols = all_of(independent_vars), names_to = "Variable", values_to =
"Value")

correlations <- ggplot(df_long, aes(x = Value, y = target)) +
  geom_point(color = "#4f81bd") +
  facet_wrap(~ Variable, scales = "free_x", nrow = 4, ncol = 4, strip.position =
"bottom") +
  theme_minimal(base_family = "sans") +
  labs(x = "", y = expression(italic(y)), color = "Target") +
   scale_y_continuous(breaks = c(0, 1)) +  # This line sets the y-axis to only
display 0 and 1
  theme(
    text = element_text(family = "sans", size = 12),
    axis.text.x = element_text(angle = 45, hjust = 1, size = 10, face = "plain"),
    strip.background = element_blank(),
    strip.text = element_text(face = "plain", size = 8),
    strip.placement = "outside",
    strip.text.x = element_text(margin = margin(t = 10))
  )

print(correlations)
model <- lm(pass_quality_gate ~ static_analysis_usage + version_control_usage +
project_size   +   academic_year   +   online_teaching   +   collaboration_mode   +
technology_choice, data=frame)

vif_results <- vif(model)

print(vif_results)

vif_df <- data.frame(Variable = names(vif_results), VIF = vif_results)

vif_df <- vif_df[order(vif_df$VIF, decreasing = TRUE), ]

ggplot(vif_df, aes(x = reorder(Variable, VIF), y = VIF)) +
  geom_bar(stat = "identity", fill = "gray") +
  theme_minimal() +
  labs(
    x = "Variable",
    y = "Variance Inflation Factor") +
  geom_hline(yintercept = 5, linetype = "dashed", color = "black") +
  coord_flip() +
  theme(text = element_text(size = 12))
```

```r
# Splitting the dataset.
X <- frame[, c("static_code_analysis", "number_of_code_lines",
               "online", "team", "language")]
y <- frame[["pass_quality_gate"]]

# Standardization.
X_std <- scale(X)
X_std <- as.data.frame(scale(X))

# Fitting Logistic Regression.
y <- factor(y)

# Model interpretation.
summary_logistic <- summary(model_logistic)
coefficients_summary <- summary_logistic$coefficients
odds_ratio <- exp(coefficients_summary[, "Estimate"])
conf_int <- confint(model_logistic)
combined_stats <- cbind(coefficients_summary, `2.5 %` = conf_int[,1], `97.5 %` =
conf_int[,2],
                        `Odds Ratio` = odds_ratio)
results_table <- round(combined_stats, 4)
print(results_table)

# Decision Tree.

frame$y <- factor(frame$pass_quality_gate)

# Fit the decision tree model
model_tree <- rpart(y ~ static_code_analysis + number_of_code_lines + online + team
+ language,
                    data = frame, method = "class")

# Print the summary of the tree model
print(summary(model_tree))

if        (!requireNamespace("rpart.plot",        quietly        =        TRUE))
install.packages("rpart.plot")
library(rpart.plot)

# Plot the decision tree
rpart.plot(model_tree, type = 4, extra = 102)

print(model_tree$variable.importance)
```

```
# Inputs importance (Permutation Importance and Cumulative Importance).
set.seed(123)
selected_features <- c("static_code_analysis", "number_of_code_lines",
                       "development_tools")
num_shuffles <- 100
baseline_preds <- predict(model_logistic, type = "response")
baseline_perf <- auc(actual = frame$pass_quality_gate, predicted = baseline_preds)

perm_importances     <-     matrix(0,     nrow     =     num_shuffles,     ncol     =
length(selected_features),
                           dimnames = list(NULL, selected_features))

for (i in 1:num_shuffles) {
  for (feature in selected_features) {
    df_shuffled <- frame
    df_shuffled[[feature]] <- sample(frame[[feature]])

     shuffled_preds  <-  predict(model_logistic,  newdata  =  df_shuffled,  type  =
"response")

     shuffled_perf  <-  auc(actual  =  df_shuffled$pass_quality_gate,  predicted  =
shuffled_preds)

    perm_importances[i, feature] <- baseline_perf - shuffled_perf
  }
}

feature_importance_means <- apply(perm_importances, 2, mean)
feature_importance_sds <- apply(perm_importances, 2, sd)

feature_importance_df <- data.frame(
  MeanImportance = feature_importance_means,
  StdDev = feature_importance_sds
)

feature_importance_df                                                              <-
feature_importance_df[order(feature_importance_df$MeanImportance,   decreasing   =
TRUE), ]

print(feature_importance_df)
```

```r
# Hypothesis Testing: Impact of Quality of Code on Students Success (grade and attempts).
test_grade <- wilcox.test(grade ~ pass_quality_gate, data = frame)
test_attempts <- wilcox.test(exam_attempts ~ pass_quality_gate, data = frame)
print(test_grade)
print(test_attempts)

results <- matrix(nrow = ncol(frame), ncol = ncol(frame), dimnames = list(colnames(frame), colnames(frame)))
p_values <- matrix(nrow = ncol(frame), ncol = ncol(frame), dimnames = list(colnames(frame), colnames(frame)))

# Loop through each pair of variables
for (i in seq_len(ncol(frame))) {
  for (j in seq_len(ncol(frame))) {
    if (i <= j) { # Avoid redundant calculations
      test <- cor.test(frame[[i]], frame[[j]], method = "kendall")
      results[i, j] <- test$estimate
      p_values[i, j] <- test$p.value
      if (i != j) {
        results[j, i] <- test$estimate # Symmetric
        p_values[j, i] <- test$p.value # Symmetric
      }
    }
  }
}

# Display the results
print("Kendall Tau correlation coefficients:")
print(results)
print("P-values for Kendall Tau correlations:")
print(p_values)

library(caret)       # For data splitting
library(glmnet)      # For logistic regression model

# Fit logistic regression model
model_logistic <- glm(pass_quality_gate ~ static_analysis_usage + technology_choice + project_size, data = frame, family = "binomial")
pd_interaction <- partial(model_logistic, pred.var = c("technology_choice", "project_size"), chull = TRUE, grid.resolution = 10)
```

k

```
# Plot
ggplot(pd_interaction, aes(x = technology_choice, y = project_size)) +
  geom_tile(aes(fill = yhat)) +  # Fill based on predicted probability
  geom_contour(aes(z = yhat), color = "white") +  # Add contour lines for clarity
   scale_fill_gradient(low  =  "white",  high  =  "darkgray",  name  =  "Predicted
Probability") +
  labs(
    x = "Technology Choice - TC",
    y = "Project Size - PS") +
  theme_minimal()


set.seed(123) # For reproducibility
data <- data.frame(
  static_code_analysis = runif(100, 0, 100),
  development_tools = runif(100, 0, 100),
  other_feature = runif(100, 0, 100),
  y = sample(0:1, 100, replace = TRUE)
)

set.seed(123) # For reproducibility
splitIndex <- createDataPartition(data$y, p = .8, list = FALSE, times = 1)
trainData <- data[splitIndex,]
testData <- data[-splitIndex,]

model_logistic <- glm(y ~ static_code_analysis + development_tools + other_feature,
data = trainData, family = "binomial")

pd_interaction  <-  partial(model_logistic,  pred.var  =  c("static_code_analysis",
"development_tools"), chull = TRUE, grid.resolution = 10)

# Visualizing interaction effects
ggplot(pd_interaction, aes(x = static_code_analysis, y = development_tools)) +
  geom_tile(aes(fill = yhat)) +  # Fill based on predicted probability
  geom_contour(aes(z = yhat), color = "white") +  # Add contour lines for clarity
   scale_fill_gradient(low  =  "#4f81bd",  high  =  "lightblue",  name  =  "Predicted
Probability") +
  labs(title = "Interaction Effect of Static Code Analysis and Development Tools",
       x = "development_tools",
       y = "static_code_analysis") +
  theme_minimal()
```

Прилог Г – Опсервације за валидацију резултата

Успех студента	Квалитет кôда						Опис пројекта		
ОС	**Б**	**Д(%)**	**Р**	***CS***	**БП**	**АГ**	**ВП**	**ТС**	**ИТ**
9	138	21,1	5	1100	29	III	67000	poj.	*wpf*
8	293	19,9	37	1700	36	III	71000	poj.	*wpf*
8	490	10,3	32	2900	39	III	143000	poj.	*wpf*
7	644	10,8	59	3400	44	III	141000	poj.	*wpf*
9	670	15,7	28	2300	29	III	89000	poj.	*wpf*
9	274	19	3	2200	45	III	75000	poj.	*wpf*
9	369	12,6	55	1800	33	III	126000	poj.	*wpf*
8	640	12,3	0	2300	26	III	133000	poj.	*wpf*
9	440	15,2	0	3300	37	III	134000	poj.	*wpf*
9	670	15,7	28	2300	29	III	89000	poj.	*wpf*
8	640	12,3	0	2300	26	III	133000	poj.	*wpf*
8	640	12,3	0	2300	26	III	133000	poj.	*wpf*
8	670	15,7	28	2300	29	III	89000	poj.	*wpf*

6	3	0	0	13	3	II	359	тимски	*angular/vue*
9	0	0	0	9	3	II	657	тимски	*angular/vue*
7	5	14,5	0	3	5	II	486	тимски	*angular/vue*
7	0	0	0	1	4	II	545	тимски	*angular/vue*
6	2	2,2	2	18	5	II	514	тимски	*angular/vue*
10	6	4,8	1	31	8	II	1200	тимски	*angular/vue*
10	0	0	0	8	0	II	455	тимски	*angular/vue*
9	0	0	0	1	3	II	372	тимски	*angular/vue*
7	0	0	0	1	0	II	196	тимски	*angular/vue*
6	0	13,8	0	9	0	II	277	тимски	*angular/vue*
7	0	0	0	1	4	II	262	тимски	*angular/vue*
6	1	0	0	4	4	II	285	тимски	*angular/vue*
7	0	0	0	20	3	II	1200	тимски	*angular/vue*
6	1	0	0	2	0	II	165	тимски	*angular/vue*
9	2	0	0	29	1	II	566	тимски	*angular/vue*
8	49	11,1	0	14	33	II	6500	тимски	*angular/vue*
6	3	0	0	13	3	II	359	тимски	*angular/vue*

6	0	0	0	5	0	IV	281	poj.	*vue*
9	0	0	0	1	0	IV	522	poj.	*vue*
8	9	0	0	10	0	IV	533	poj.	*vue*
9	0	0	0	0	0	IV	244	poj.	*vue*
10	4	0	0	3	2	IV	972	poj.	*vue*
9	0	0	0	1	0	IV	417	poj.	*vue*
9	0	0	0	1	0	IV	466	poj.	*vue*
8	12	0	0	5	4	IV	586	poj.	*vue*
7	0	0	0	0	2	IV	361	poj.	*vue*
8	0	0	0	2	0	IV	387	poj.	*vue*
10	1	0	0	1	0	IV	326	poj.	*vue*
9	0	0	0	1	0	IV	285	poj.	*vue*
6	0	0	0	5	0	IV	281	poj.	*vue*
9	0	0	0	1	0	IV	522	poj.	*vue*
8	9	0	0	10	0	IV	533	poj.	*vue*

*поj. – појединачно

План третмана података

Назив пројекта/истраживања
Модел оцењивања софтверских пројеката заснован на статичкој анализи кôда
Назив институције/институција у оквиру којих се спроводи истраживање
а) Универзитет у Новом Саду, Факултет техничких наука б) в)
Назив програма у оквиру ког се реализује истраживање
-
1. Опис података
1.1 Врста студије *Укратко описати тип студије у оквиру које се подаци прикупљају* Студија спроведена у оквиру дисертације је обухватала квантитативне истраживање са циљем тестирања постављених хипотеза и верификације предложеног модела. 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 мерење квалитета кôда софтверских пројеката и прикупљање информација о поставкама пројеката и академском успеху студената

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 ___ИМ1 – 13, ИМ2 – 6_____

б) број мерења (испитаника, процена, снимака и сл.) ИМ1 – 710, ИМ2 - 121_

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

Методологија за прикупљање података подразумевала је бележење резултата спровођења статичке анализе кôда над софтверским пројектима, бележење података о поставци и организацији пројекта на различитим предметима. Такође, у склопу ИМ2 прикупљање података вршено је путем упитника.

2.1.1. У оквиру ког истраживачког нацрта су подаци прикупљени?

а) експеримент, навести тип _____

б) корелационо истраживање, навести тип _____ИМ2 - Истраживање путем упитника __

ц) анализа текста, навести тип __ИМ1 – Статичка анализа софтверског кôда_____

д) остало, навести шта _____

2.1.2 Навести врсте мерних инструмената или стандарде података специфичних за одређену научну дисциплину (ако постоје).

ИМ1 – алат за статичку анализу кôда – *SonarQube, ИМ2* – Упитник за прикупљање података, коришћена је Ликертова скала мерења_____

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

_____ИМ1 – пажљивим избором алата за статичку анализу кôда, према искуствима из прегледа стања у области, ИМ2 – укључена су питања из већ постојећих и валидираних скала, што је допринело поузданости и валидности прикупљених података_____

2.2.3. На који начин је извршена контрола уноса података у матрицу?

_____ИМ1 – подаци добијени из алата за статичку анализу кôда су аутоматски извезени у формате који су компатибилни са програмима за обраду података, ИМ2 - упитик је спроведен путем онлајн платформи, што је омогућило директан унос одговора учесника у дигиталну базу података и смањило могућност грешака при ручном уносу.

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**

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.

Данило Николић (2024): Модел оцењивања софтверских пројеката заснован на статичкој анализи кôда

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

Данило Николић, nikolic.danilo@uns.ac.rs

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

Данило Николић, nikolic.danilo@uns.ac.rs

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

Данило Николић, nikolic.danilo@uns.ac.rs