Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2022.Doi Number

Uncovering Determinants of Code Quality In Education Via Static Code Analysis

Danilo Nikolić[1], (Graduate Student Member, IEEE), Darko Stefanović[1], (Member, IEEE), Miroslav Nikolić[2], (Graduate Student Member, IEEE), Dušanka Dakić[1], (Member, IEEE), Miroslav Stefanović[1], (Member, IEEE), Sara Koprivica (Graduate Student Member, IEEE)

[1]Faculty of Technical Sciences, University of Novi Sad, 21000 Novi Sad, Serbia
[2]Open Institute of Technology, University of Malta, XBX 1425, Malta

Corresponding author: Darko Stefanović (e-mail: darko.stefanovic@uns.ac.rs).

ABSTRACT The role of static code analysis in enhancing the quality of software codes is widely acknowledged. Static code analysis facilitates the examination of code for irregularities without program execution, which significantly impacts project quality. Furthermore, tools for static code analysis serve as educational aids, imparting essential lessons on coding practices. Motivated by the growing complexity of software projects and the pivotal role of code quality in academic performance within computing disciplines, this research examines over 500 student projects using static code analysis tools. The aim is to determine metrics that influence the code quality of student projects. The study investigates how metrics, such as project setup, influence code quality and students' academic performances. By adopting a broad approach, the investigation determines the overall impact of these metrics on the technical integrity of software engineering projects and academic outcomes. Insights derived from this study are anticipated to enhance teaching strategies and curriculum development, aiming to improve academic performance by promoting better code quality.

INDEX TERMS academic performance, code quality, education, educational practices, static code analysis

I. INTRODUCTION

The quality of source code is a fundamental aspect of any software product, necessitating continuous verification and monitoring to ensure its robustness, efficiency, and maintainability. Static code analysis plays a main role in this process of continuous improvement [1]. This method was created in the early 1960s to enhance compiler operations by evaluating source code without executing it [2]. The primary objective of static code analyses is to detect potential bugs, vulnerabilities, and unwanted patterns within the code that may lead to problems in software functionality or its security. Using this analysis helps find and fix problems early before the software moves on to the testing or production stages [1].

Since then different tools for static code analysis have been developed and over the years expanded their application to debugging tools and software development frameworks, marking a significant evolution from its inception [1], [3].

The evolution of static analysis tools from basic lexical analyzers to sophisticated systems that can examine code for a wide range of programming languages highlights their essential role in maintaining and improving code quality [3]. These tools are capable of identifying deviations from defined quality standards without making automatic modifications to the code itself, demonstrating their extensive utility in ensuring the robustness, efficiency, and maintainability of software products. A growing number of tools enable static analysis of code written in various programming languages, generating reports and highlighting deviations from defined code quality standards [3], [4]. As modern software systems continue to grow and evolve, the need for such analysis tools becomes increasingly significant. These tools are finding their place not only in the industry but also in education, where they are being applied to projects developed by future software engineers [5], [6].

Within the scope of this research, static code analysis serves as a tool for the objective evaluation of the quality of student software projects, thereby allowing for further examination of their correlation with academic performance.

The foundation of this research is in the understanding that the quality of software code can significantly impact the success of software projects [2], [7], and thereby, the educational outcomes achieved by students in the domain of software engineering [8].

The emphasis on clearly defined software quality standards [9], guides the focus toward the need for a tool that provides quantitative metrics of software quality. In this paper, SonarQube [10] has been used for retrieving such metrics. SonarQube is an open-source tool designed for reviewing code quality by conducting static code analysis. This approach supports the theory that, through the proper measurement and analysis of quality metrics [7], the technical value of software projects can be enhanced, thereby enriching the knowledge acquired by students.

This research aims to investigate and analyze the correlation between code quality metrics obtained through the implementation of static code analysis using the SonarQube tool and course grades and the time taken for passing projects. The study also examines how various factors, including the mode of teaching (online versus traditional approach), choice of technologies, collaboration mode (team-based or individual), the academic year in which the project was undertaken, project size, as well as the use of version control tools and static code analysis tools, influence the quality of code generated by students.

Within the framework of the study, quantitative research was conducted to analyze the correlation between code quality metrics and students' academic performances in the field of software engineering, utilizing a random sample of over 500 student projects. The code quality metrics analyzed in the study include duplications, security hotspots, vulnerabilities, bugs, and code smells, which are generated by the selected tool for conducting the analysis. This approach allows for a comprehensive investigation of the impact of various factors on learning outcomes in the domain of software engineering. The study employed a variety of statistical analysis methods, including Kendall's rank correlation, Binomial Logistic Regression, and the Mann-Whitney U test, to uncover the correlations between project setup variables, code quality, and student academic performance.

The contribution of this paper is to enrich the existing knowledge in the field of software engineering education. Through a detailed analysis of static code analysis metrics, the study identifies key factors that contribute to high-quality student software projects. The use of static code analysis tools and the choice of technology emerged as significant positive influences, whereas larger project sizes were found to have a negative correlation with code quality. Additionally, the results include the effects of other variables, observed within the research, on code quality. Furthermore, a clear correlation between code quality and the academic performance achieved by students is presented and described.

The expectation is that this research will lay the groundwork for the development of improved strategies for teaching software engineering, which should result in a generation of software engineers ready to face the challenges of contemporary software development.

The findings presented in the study are intended to provide educators, students, and professionals in software engineering with deeper insights into how learning approaches, technology choices, and collaboration can affect code quality and, consequently, students' academic performances. This study provides significant insights into the impact of various project setup metrics on code quality and academic performance in software engineering education. By employing a multi-dimensional analysis, we bridge the gap between educational practices and software engineering standards, offering actionable insights for curriculum designers and educators to enhance academic outcomes.

Beyond the introduction and conclusion, the paper is organized as follows: Section II provides a literature review in the field of static code analysis, highlighting the importance of applying static analysis tools in the industry, as well as their potential and methods of application in education. Section III describes the procedure of static code analysis tool selection, as well as the chosen metrics for research. Pre-processing of the data and the statistical methods that are used are represented in Section IV. Section V presents the results obtained from the conducted research, and Section VI discusses these results and establishes relationships between the observed variables and code quality metrics. This section also outlines some limitations of the research, providing a comprehensive overview of the state of static code analysis in both the professional and educational realms.

II. LITERATURE REVIEW

With the advancement of static code analysis tools, there has been a significant increase in scholarly research in the area of static code analysis. As part of prior work in the field, a literature review was conducted [4], highlighting the most commonly used tools in scientific research, and the programming languages they support, among other aspects. On the other hand, some research often focuses on newly developed static code analysis tools. A study [11] introduces a tool for the static analysis of Programmable Logic Controller (PLC) program code, detailing its application and the results it achieves in industrial projects.

An empirical study illustrating the use of static code analysis in the industry and the willingness of developers of varying seniority levels to utilize these tools is described in [12]. The same paper also demonstrates that these tools are increasingly being used as a mandatory part of the deployment pipeline in the industry. The study [13] underscores the necessity for a deeper understanding and measurement of software quality beyond traditional metrics. Additionally, this study compares programming habits between professionals in the industry and students, highlighting areas where students are more prone to making mistakes that could compromise the code quality.

Furthermore, static code analysis tools are increasingly finding their application in education. Research [5] presents the CodeMaster tool based on static code analysis, which automatically grades student projects, emphasizing the importance of such tools in enhancing the quality of education for future IT engineers. Similarly, another study [14] discusses the use of tools for the automatic verification of programming tasks, underlining their significance in educational applications.

In [15], the results of an experiment are shown where students independently utilized the Programming Mistake Detector (PMD) static code analysis tool to correct irregularities in their projects. The results of this study also reveal that students largely understand the reports generated by this static code analysis tool and have the ability to amend these irregularities.

Another study [16] incorporated a static code analysis tool into the educational process in a manner that a portion of the students used the tool during their project work, while a control group did not use the tool for their projects. The results indicate that students who used the static code analysis tool achieved better outcomes and developed higher-quality source code compared to the control group.

In [17] the development and implementation of the Edgar system for the automatic evaluation of software projects is described. This system provides an objective and efficient way to assess a large number of student projects, as well as timely and useful feedback to students. Within their study, the importance of this system's objectivity in the process of evaluating student projects as opposed to traditional grading methods is emphasized, showcasing an example and results of using the Edgar tool in an educational system.

Authors of [18] underscores the importance of incorporating static code analysis tools into the educational process. This research also speaks to the need for introducing new metrics to be used in analyzing student projects in comparison to industry projects. Additionally, the paper addresses various impacts on code quality among students, with research [19] highlighting the significance and impact of teamwork on the quality of student projects. This investigation is an example of the effective application of these tools in the educational process, especially emphasizing the ability to identify students who contribute less during teamwork and how this reflects on the code quality. Another study [20], that discusses the impact of teamwork on project outcomes, also talks about the challenges of creating group projects, the influence of project size on outcomes, and the results and methods of evaluating such projects.

Beyond the impact of teamwork on student project outcomes, another variable examined is the choice of project technology. Research [21] shows that the choice of software technologies can significantly impact the quality of the source code. Therefore, this research investigates how the selection of specific technologies influences project success and code quality.

Furthermore, the study [22] explores the impact of functional and object-oriented paradigms on code quality, presenting significant differences in terms of specific code quality metrics in its findings. In a similar vein, research [23] yields results that lead to the conclusion that utilizing the C++ programming language instead of C can lead to enhancements in software quality, reduction in complexity, decreased error proneness, and lower maintenance effort.

Although research [24] discusses the differing outcomes achieved by students in an online education environment compared to traditional classroom instruction, this literature review did not identify any studies discussing the direct impact of different teaching methods on code quality in student projects.

As a determinant of code quality, the use of versioning tools will also be considered. In this context, the study [25] demonstrates how the use of versioning tools using Bitbucket [26] contributes to the understanding and application of coding standards. Conversely, research [27] highlights that utilizing versioning tools using Github [28] facilitates the implementation of static code analysis, thereby ensuring a higher quality of the project's source code.

The experience of software engineers is measured in various ways as a metric in the industry. Studies [29], [30] show how the experience of software engineers affects the quality of code they produce during development.

Research [31] analyzes how the number of lines of code affects the complexity of software and concludes that an increase in the number of lines of code in a software project leads to increased complexity. This, in turn, affects the increase in the number of errors, as well as maintenance difficulties, reliability, and software performance.

A special category of literature included in this review focuses on methods of measuring student performances in various areas. Thus, [24], [32], [33] consider the grade a student achieves in a course as an indicator of academic performance upon completion of that course. On the other hand, another variable indicating student performance is whether the student fulfills their obligations on time. Studies [24], [34] also take the metric of time or adherence to deadlines in terms of fulfilling obligations as a variable indicating the performance of the student.

Ways of applying static code analysis tools are analyzed in [35], [36]. These researches identify different strategies for using static code analysis tools on projects. The first identified strategy in the research [35] entails the continuous application of static code analysis tools on the source code of a project during the software development process, whereas the second strategy involves applying tools immediately after software development is completed. These strategies have been evaluated within the research [36], presenting the advantages and disadvantages of their application.

The review paper [37], which explores the software obfuscation metric through a systematic literature review, highlights the importance of this metric for code quality, as well as the need for standardization of this metric. Such standardization would facilitate its incorporation into static code analysis tools and enable broader studies on industrial projects to be conducted.

In summary, the importance of using static code analysis tools is reflected in the increasing need to enhance code quality in industrial projects [11], [12]. Following closely behind the industry, the significance of improving code quality has begun to be applied in the education of future software engineers in various ways. Some research [18] shows examples where students independently apply these tools to their projects to correct the irregularities they find, whereas on the other hand, certain studies [5], [17] suggest systems and tools that enable educators to evaluate student projects most efficiently and objectively possible. Our study, on the other hand, aims to explore different factors in the setup of student projects that can affect the quality of the source code. This review encompasses research that analyzes these factors and places them in the context of educating future software engineers. Thus, study [19] discusses the factor of team and individual projects, then study [21] highlights the factor of technology choice, followed by [16] the use of static code analysis tools during project work, as well as the teaching mode [24]. In addition to the impact of these and other factors on code quality, this study will also measure whether code quality can be correlated with the student's performance in their courses, in terms of the grade obtained and the speed of passing the project [33], [34].

The synthesis of the results obtained from the analysis of the studies covered in the literature review is presented in Table I.

TABLE I
SYNTHESIS OF LITERATURE REVIEW

Topic		Paper
Static code analysis tools in the industry		[3], [11], [12]
Static code analysis tools in education	Used by professors	[5], [13], [17]
	Used by students	[14], [15], [16]
Determents of code quality	Collaboration mode	[19], [20]
	Technology choice	[21]
	Project size	[31]
	Experience	[29], [30]
	Version control usage	[25], [27]
	Static analysis usage	[16]
	Teaching method	[24]

While previous studies have explored individual factors affecting code quality, this study uniquely integrates multiple dimensions, to provide a comprehensive understanding of their combined impact on both code quality and academic performance. Additionaly, study [38] focuses on the impact of code quality and submission behavior on teaching strategies but does not consider other project setup variables. Research [39] examines learning behavior and code quality improvement in an automated programming assessment environment, focusing on individual learning behaviors without considering broader project setup factors. Study [40] analyzes the gap between the software industry and software engineering education, primarily focusing on curriculum alignment without delving into the combined impact of project setup variables on code quality.

This literature review highlights the diversity of tools and approaches in static code analysis, making it evident that the focus of most research in this area is directed toward two main aspects: enhancing the technical quality of software projects and measuring the impact of these tools' applications for educational purposes. However, few studies directly explore the impact of various aspects on code quality and how, if at all, the generated code quality affects the academic performance of future software engineers, leaving room for further research in this area.

This study seeks to bridge this gap by exploring how different factors affect the efficacy of static code analysis in an educational setting. The aim is to expand the current understanding of these factors, using empirical evidence to support or refute the existing findings from the literature.

Furthermore, there is significant interest in the community for the application and evaluation of static code analysis in both industry and education. However, there is substantial room for additional research that could provide deeper insights into how different pedagogical and technical interventions may impact code quality and whether there is a clear correlation between code quality and students' academic performance. This study addresses these questions.

Unlike previous research, this research uniquely quantifies the influence of project setup variables on code quality and examines their direct correlation with academic performance metrics. Additionally, this study employs a broader, multi-dimensional analysis that integrates various educational contexts, providing a comprehensive understanding of static code analysis's role in enhancing software engineering education.

III. SELECTION OF STATIC CODE ANALYSIS TOOL AND PRESENTATION OF CHOSEN METRICS

In section II of this study, various research are mentioned where different tools for static code analysis were utilized on software projects. However, for this study, a single tool will be used to analyze all projects included in the research.

The number of tools available for static code analysis is substantial, yet, as described in the study [4], tools such as CppCheck [41], FindBugs [42], SonarQube [10], PMD[43], and Splint [44] are most frequently used in research.

A. SELECTION OF TOOL FOR STATIC CODE ANALYSIS

Some of the tools for static code analysis enable analysis of source codes written in a single programming language, whereas others provide support for conducting analyses in various programming languages [3]. Given that the analysis in this study will include software projects whose source code is written in different programming languages, the first criterion (C1) for tool selection includes only those tools that offer support for all programming languages used in the projects involved in this research.

The second criterion (C2) for tool selection is the availability of a free version that provides quality analyses of the code. Primarily due to financial constraints for conducting the research.

Finally, as a third criterion (C3), results obtained from a study [45] that uses the Determining an Evaluation Methodology for Software Methods and Tools (DESMET) [46] for evaluating static code analysis tools were utilized. Results from [45] represent SonarQube as the best tool (with a score over 69%) for conducting static code analysis by DESMET evaluation.

By applying these criteria, the tool chosen for static code analysis is SonarQube, which will be used in conducting the research. This tool offers support for all programming languages included in this study, allows for the analysis of metrics that will be used in the research in its free version, and finally, has received the highest rating within [45] according to the DESMET methodology of evaluation. The presentation of tools and their compliance with the specified criteria is shown in Table II.

TABLE II
SELECTION OF STATIC CODE ANALYSIS TOOL

Tool	C1	C2	C3
SonarQube	✓	✓	✓
FindBugs	✕	✓	✕
PMD	✕	✓	✕
CppCheck	✕	✕	✕
Splint	✕	✓	✕

SonarQube is an open-source tool developed by SonarSource for continuous code quality review. It conducts static analysis of the code, providing detailed reports on bugs, code smells, vulnerabilities, security hotspots, and code duplications [47]. Research [48] has demonstrated SonarQube's effectiveness as a valuable tool for identifying various types of code irregularities. Beyond irregularity identification, SonarQube offers additional benefits including effort estimation, detailed problem descriptions, and guidance for modifying source code to resolve problems, thereby positively affecting productivity in software development [49]. Furthermore, numerous studies have been conducted to verify the accuracy of the reports generated by SonarQube, reinforcing its importance and credibility [45], [50].

B. CODE QUALITY METRICS

According to the software product code quality standard ISO/IEC 25010 [9], eight quality categories of code are defined as follows:

1. Functional Suitability,
2. Performance Efficiency,
3. Compatibility,
4. Interaction Capability,
5. Reliability,
6. Security,
7. Maintainability,
8. Flexibility,
9. Safety.

The selected tool, SonarQube, addresses the categories of reliability, maintainability, and security from the ISO/IEC 25010 standard within its basic set of code quality rules [51]. Other categories from [9] are less emphasized as they often require domain-specific knowledge and context that goes beyond automated static code analysis.

In addition to these categories, it also includes the category of source code complexity and emphasizes the importance of these metrics in depicting the quality level of the source code in its documentation. The subcategories of these metrics that quantitatively represent code quality and will be used in this research are [51]:

- Code duplication increases software complexity and makes maintenance more difficult, which directly affects code quality reduction. By removing duplicates, development teams can significantly improve software reliability and efficiency, facilitating updates and reducing the potential for errors [52].
- Security hotspots and vulnerabilities play a crucial role in assessing code quality, as their presence can seriously compromise the reliability and maintainability of software. Detecting and correcting these vulnerabilities not only increases software security but also contributes to an overall improvement in code quality, making it more resistant to attacks and easier to maintain. Integrating the measurement of security vulnerabilities as a code quality metric allows organizations to develop more robust and secure software products, thereby raising the overall quality standard in the software industry [53].
- Bugs are direct indicators of the quality of the software development process and are essential for maintaining software functionality [1].

- Code smells indicate critical flaws in design, implementation, or maintenance processes that could potentially degrade software quality, highlighting the need for continuous research and advanced detection methods to ensure the development of high-quality software [54], [55].

SonarQube quantitatively represents all these metrics in the reports it generates, and these results will be used in the analysis of this study. Quantitative measurement, in this case, refers to the number of recorded irregularities from a specific category found within the source code of the software project. In addition to the number of recorded irregularities, this tool also generates additional information about specific irregularities, such as severity, labeling each irregularity with a level of danger: blocker, critical, major, and minor. These additional attributes describing irregularities will not be considered during the analysis within this study.

C. PROJECT SETUP METRICS AS DETERMINANTS OF CODE QUALITY

Based on the literature review presented in section II key metrics are identified that will be analyzed as determinants of code quality in student projects within this study are presented in Table III.

TABLE III
PROJECT SETUP METRICS USED AS DETERMINANTS OF CODE QUALITY

Metric	Label	Possible values
Technology choice	TC	1. .NET 2. Spring and Angular 3. Java 4. .NET - WPF 5. .NET and Angular 6. HTML, CSS, and JS
Collaboration mode	CM	Team/Individual
Project size	PS	Number of code lines
Academic Year	AY	I, II, III, IV
Version control usage	VCU	Used/Not used
Static analysis usage	SAU	Used/Not used
Teaching method	TM	Online/Live

1) TECHNOLOGY CHOICE (TC)

Depending on the technology chosen for software development, different types of irregularities in the code may occur more frequently, significantly impacting software quality [21]. The mentioned study indicates that functional languages lead to better code quality compared to procedural languages. Similarly, another study [22] highlights the increased complexity associated with these languages. Additionally, a study [23] directly compares C and C++ programming languages, presenting results that show the use of C++ reduces complexity, as well as the number of other irregularities. Based on these studies, our research considers the choice of technology as a determinant of code quality, assuming that the technology choice can influence the outcome of code quality in student projects.

2) COLLABORATION MODE (CM)

Another metric used in this research as a determinant of code quality is the collaboration mode, whether students organized their projects in teams or developed their projects individually. A certain number of projects included in this research consists of projects developed individually, whereas another group includes team projects. Previous studies [19], [20] emphasize the impact of team projects on code quality outcomes. However, the results of this research will display differences in code quality outcomes between individual and team-organized projects.

3) PROJECT SIZE (PS)

Indicators in the industry suggest that code quality decreases as project size increases [31]. The number of lines of code developed in the project is taken as an indicator of project size in this research. Through these analysis results, we aim to demonstrate whether students make bigger mistakes when working on larger projects compared to the code quality analysis results on smaller projects. This aims to answer whether project size is a significant determinant of code quality in software engineering education.

4) ACADEMIC YEAR (AY)

It has been shown that the experience of software developers in the industry plays a significant role in the quality of code they generate [29], [30]. These studies list various metrics for measuring developer experience. In this research, the academic year in which the student works on a specific project is taken as an indicator of the developer's experience. This measure of experience reflects the experience students have gained in understanding software development.

5) VERSION CONTROL USAGE (VCU)

The use of version control tools is considered one of the possible determinants of code quality. The research [25] included in the literature review in section II of this paper emphasizes the significance of using version control tools like Bitbucket and implementing peer code review processes in student projects. The study also shows that the use of these tools contributes to the understanding and application of coding standards, identification of irregularities, and improvement of coding skills among students. Our research includes student projects that used version control tools and those that did not. This way, this paper aims to demonstrate whether and to what extent the use of version control tools contributes to higher code quality.

6) STATIC ANALYSIS USAGE (SAU)

Section II of this paper described studies [15], [16] that discuss the results of using static code analysis tools in the teaching process. This research will cover student projects within courses where the use of static analysis tools was mandatory, as well as projects developed without these tools. This will not only show whether the use of these tools achieves higher code quality in software projects but will also provide a clearer picture of students' abilities to handle static analysis tools, interpret the results, and rectify detected irregularities.

7) TEACHING METHOD (TM)

Due to the Covid-19 classes were mainly held online during the one semester of the 2019/2020 academic year and also throughout the 2020/2021 academic year. Student projects from these years were used for this research to show the results students achieve in terms of code quality in an online teaching mode. After these academic years, classes returned to the traditional mode of teaching, i.e., held in person at the University premises. The second part of the projects used in this research involves projects that were carried out when the faculty's teaching was conducted in a traditional mode. Numerous studies, such as [24], examine the impact of online teaching on the results students achieve. However, there are still no studies examining the impact of this teaching mode on the quality of code generated by students. This research will present the results of analyzing projects carried out in both mentioned teaching modes and discuss the determinacy of the teaching method on code quality.

D. METRICS AS DETERMINANTS OF ACADEMIC PERFORMANCE

In addition to the metrics used as determinants of code quality, this research also establishes academic performance metrics, for which the correlation with the code quality of student projects has been examined. The academic performance metrics set are the grades students achieve in the course within which the analyzed project was carried out, as well as the number of attempts needed for a student to complete the project.

The metrics of academic performance are represented in Table IV.

TABLE IV
METRICS USED AS DETERMINANTS OF ACADEMIC PERFORMANCE

Metric	Label	Possible values
Grade	G	NP, 6, 7, 8, 9, 10
Project completion timeframe	PCT	>0

1) GRADE (G)

In many research studies [32], [33], grades are considered an indicator of performance at all levels of education.

The first academic performance metric in higher education chosen for this research is the grade a student achieved in the course for which the project, subjected to static code analysis, was carried out. Possible values for the grade metric range from 6 to 10, as well as the 'NP' (Not Passed) mark, which is assigned to students who have not yet completed the course. These grades are defined according to the standards of the Faculty of Technical Sciences, where the observations were recorded, and were taken from courses held in previous years, with grades assigned by the course instructors.

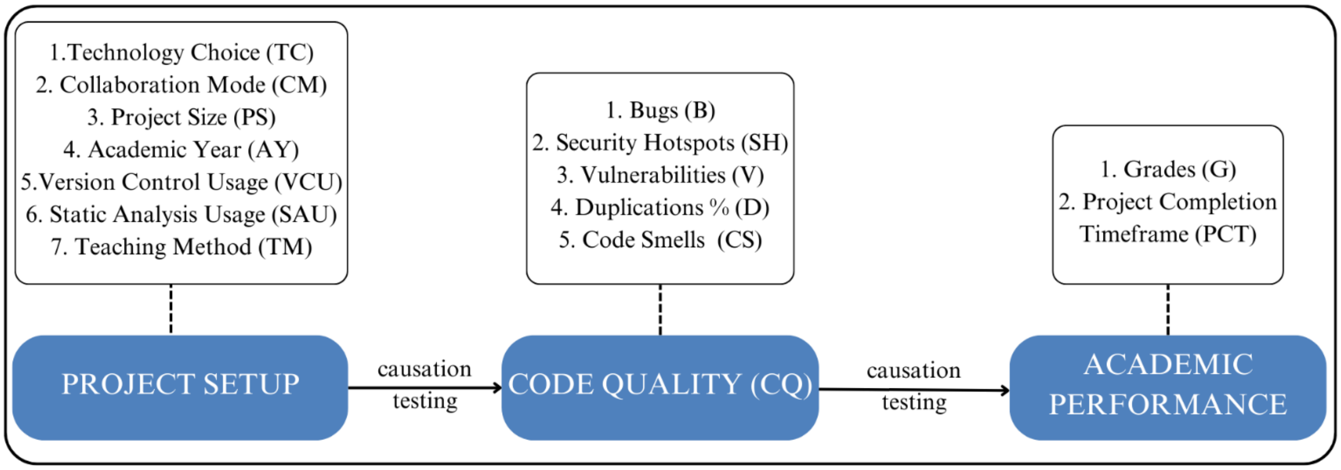

FIGURE 1. Research framework

The grading scale used ranges from 6 to 10, where 6 represents the lowest possible passing grade, and 10 represents the highest possible grade. The goal of monitoring this metric is to analyze the relationship between the results of code quality and the grade a student achieves in the course.

2) PROJECT COMPLETION TIMEFRAME (PCT)

In addition to grades, the Project Completion Timeframe (PCT) required for a student to successfully pass the course project was also recorded as an academic performance indicator. PCT is measured by the number of attempts a student takes to pass the course project. Possible values for this metric are positive integers (>0), indicating the number of attempts, and the 'NP' value, which is assigned if the student has not yet completed the course project. The inclusion of this metric as an indicator of academic performance is supported by other studies in the field of education[34], which often consider the dimension of time along with outcomes as indicators of academic performance.

E. RESEARCH FRAMEWORK

Within the scope of this research, based on the metrics presented, an examination and analysis will be conducted to determine the extent to which a correlation exists between variables from the project setup category and the code quality. Furthermore, an examination and analysis of the correlation between code quality and the academic performance of students will also be undertaken. The research framework is presented in Fig. 1.

IV. METHODOLOGY

Data were collected by observing various project setups across different courses from different academic years, according to seven variables outlined in Section III. Subsequently, analyses of these student projects were conducted using the SonarQube tool, with results from each individual project being recorded alongside the academic outcomes students achieved in the course.

A. DATA PRE-PROCESSING

The dataset examined was devoid of missing values or duplicated rows. It comprised nominal variables with binary outcomes (e.g., SAU, VCU, TM, and CM) where 1 indicated usage or a specific condition being met, and 0 represented its absence. Specifically:

- SAU and VCU were coded 1 when utilized and 0 otherwise,
- TM was 1 for online lecture attendance and 0 for live attendance, and,
- CM was 1 when students collaborated in teams and 0 when they worked individually.

In addition to binary nominal variables, it comprises a nominal variable with 6 values, i.e. TC. It has the following values:

- .NET and Angular (130 occurrences),
- .NET (126 occurrences),
- HTML/CSS/JS (74 occurrences),
- Java and WPF (62 occurrences each),
- Spring and Angular (52 occurrences).

The dataset also included variables measured on continuous scales, such as:

- bugs (range: 0 to 201),
- duplications % (range: 0 to 45.9),
- code smells (range: 0 to 2600),
- security hotspots (range: 0 to 92),
- PS (range: 64 to 3200)

Finally, there are 4 ordinal variables: AY, G, PCT, and vulnerabilities.

As it is shown in Table V and Fig. 2, all continuous variables are right-skewed. They are also on different scales, with PS having the largest scale (min: 64.0, max: 32,000) and duplications % having the smallest scale (min: 0.0, max: 45.9)

TABLE V
SUMMARY STATISTICS FOR CONTINUOUS VARIABLES

	B	CS	D(%)	PS	SH
Min.	0.0	0.0	0.0	64	0.0
1st Quartile	0.0	40.0	1.4	1,300	1.0
Median	5.0	277.0	5.0	5,250	4.0
Mean	17.2	356.7	7.2	9234	5.9
3rd Quartile	22.0	573.0	12	17,000	8.0
Max	201.0	2,600	45.9	32,000	92.0

Within the dataset's ordinal variables, the PCT is noted for having the highest count of factors, totaling 9, whereas vulnerabilities exhibit the widest range, reaching 12, as detailed in Fig. 3. With a consideration of 1 degree of freedom, the standard deviations for these variables are as follows: 1.05 for AY, 1.53 for G, 1.07 for PCT, and 3.53 for vulnerabilities.

The distribution of the binary nominal variables in the dataset is as follows: for code quality, 222 instances are marked as 0 and 284 as 1; collaboration mode shows 189 instances of 0 and 317 of 1; static analysis usage is observed with 436 instances of 0 and 70 of 1; teaching method records 417 instances of 0 and 89 of 1; version control exhibits 136 instances of 0 and 370 of 1.

During the dataset preparation, three key data pre-processing steps were employed: encoding of the TC variable, the introduction of a novel variable termed code quality (created as the interaction term), and data standardization.

1) FREQUENCY-BASED ENCODING OF TC VARIABLE

Encoding of the TC variable was performed based on the frequency of category occurrences. This was done for two reasons:

- frequency encoding converted the category frequencies into ordinal values, thereby preserving the information on the relative prevalence of each category,
- this method was chosen to avoid the dummy variable trap, which could potentially lead to an increase in the dataset's dimensionality, also known as the curse of dimensionality.

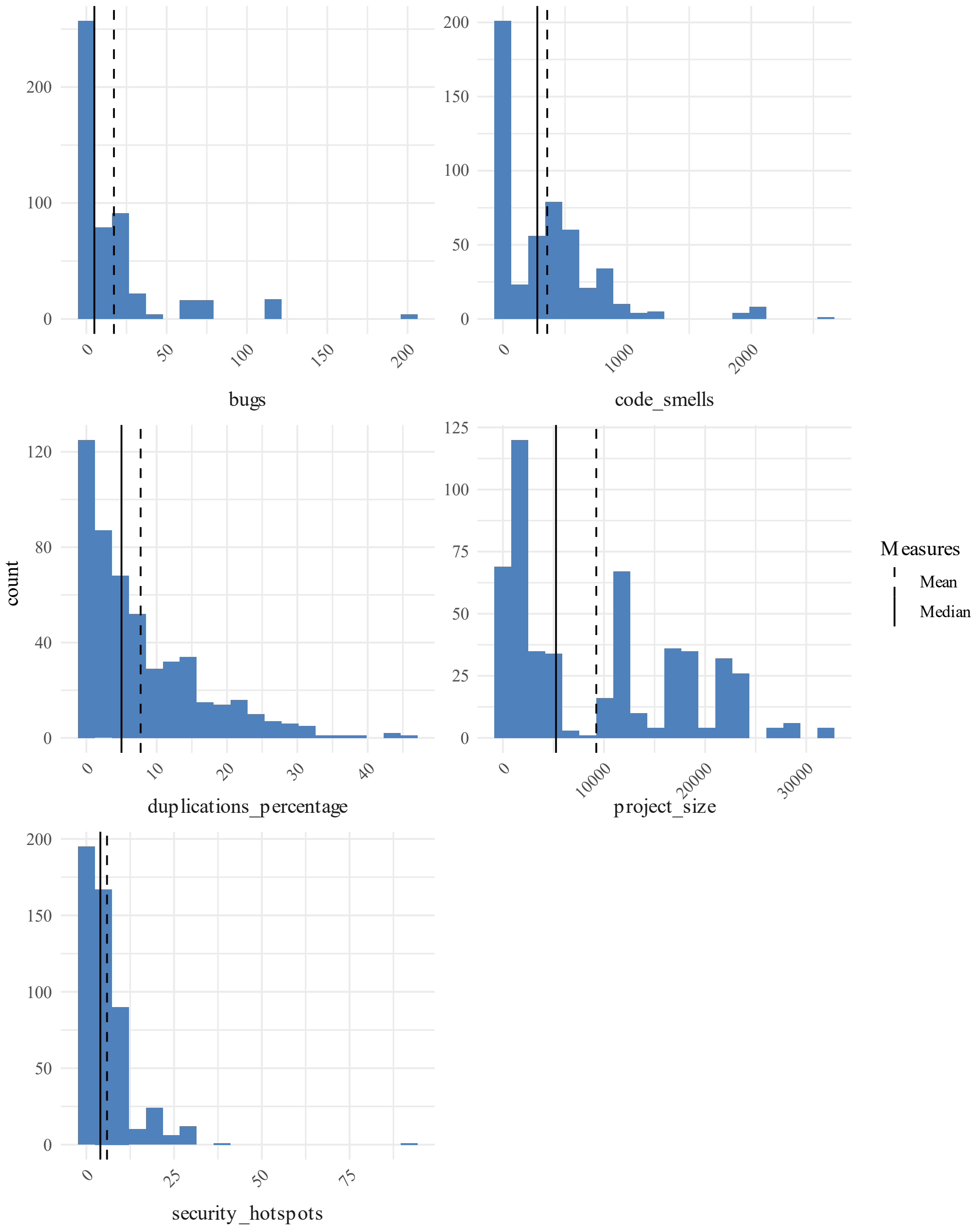

FIGURE 2. Continous variables' data distribution

2) CREATION OF NEW VARIABLE (CQ)

To investigate the association and impact of the project setup variables vector, which encompasses bugs (B), duplications % (D), vulnerabilities (V), code smells (CS), and security hotspots (SH), on code quality, a new binary variable named code quality (CQ) was developed. The binary outcomes for this variable were defined as {0,1}, with a selected threshold of 0.5 serving as the benchmark for classification. In constructing this variable, all contributing factors were accorded equal significance.

The formulation for calculating the dependent variable, CQ, is given by (1).

Weights are uniformly distributed, summing to 1 (W_i = 0.2) whereas the variable base value is its value proportion of its maximum value, for example, if the maximum range of bugs variable is 100 and the observation value is 40 then B x W_b = 0.4 x 0.2 = 0.08.

$$CQ = B\ x\ Wb + \ D\ x\ Wd \ + \ F\ x\ Wf + CS\ x\ Wcs + SH\ x\ Wsh \qquad (1)$$

The classification of an observation as 1 (indicating higher code quality) or 0 (indicating lower code quality) is determined based on whether the computed value of CQ meets or exceeds the 0.5 threshold (2).

$$\begin{cases} 0 & if\ CQ < 0.5 \\ 1 & if\ CQ \geq 0.5 \end{cases} \qquad (2)$$

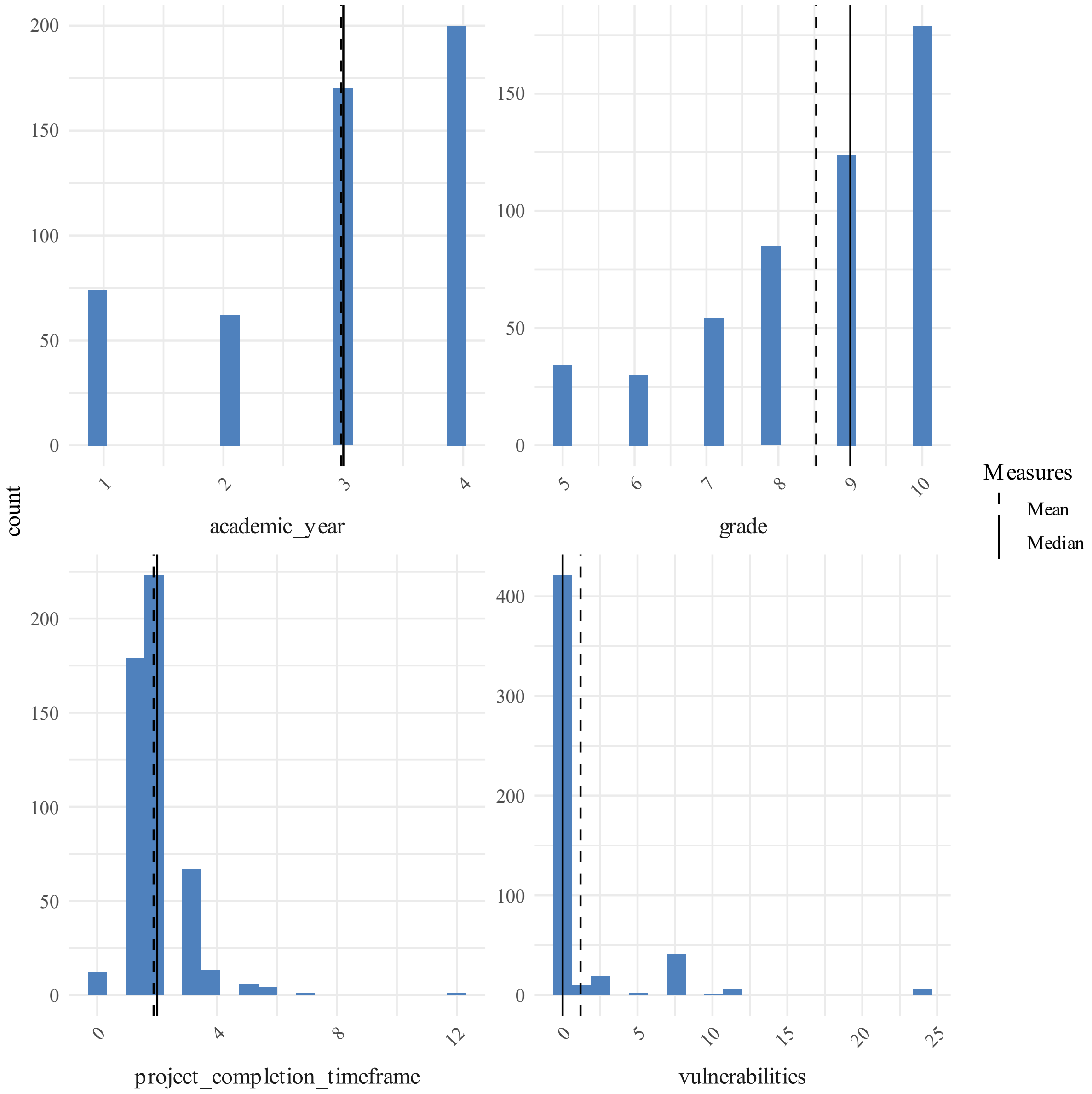

FIGURE 3. Ordinal variables' data distribution

This approach ensures that each variable contributes equally to the overall assessment of code quality, facilitating a balanced and comprehensive evaluation.

3) DATA TRANSFORMATION - STANDARDIZATION

In order to fit the statistical analysis technique to test the impact of the project setup on the code quality, it is needed to conduct data transformation and standardize existing scales to the variance of 1 and mean of 0 $\{(X_i\text{-}X\mathit{mean})/X_{\text{std}}\}$

B. ANALYSIS - Impact of project setup on the code quality

1) CORRELATIONS BETWEEN VARIABLES

Associations between all project setup variables and CQ, totaling five, were explored using the Kendall rank correlation coefficient (Table VI). This coefficient was selected following the identification of a monotonic relationship among non-normally distributed data. To address multicollinearity, the variance inflation factor (VIF) was employed, adopting a threshold of 5, as depicted in Fig. 4. Given that Logistic Regression requires independence of inputs, the interpretation of each input's coefficient/slope is conducted with the assumption that other inputs are held constant. A VIF of 5 is generally considered the threshold beyond which the precision of coefficient estimations could be significantly compromised [56].

2) CAUSATION AND EFFECT SIZE

In examining the effects of project setup variables on CQ, Binomial Logistic Regression was utilized, aiming for a 5% significance level to ascertain statistically significant impacts. To enhance the interpretability of effect size, odds ratios were included in the analysis, calculated as the exponential of the coefficients (OR=e), and presented in Table VII. Binomial logistic regression, a statistical technique for classifying the effects of independent variables on a binary categorical dependent variable, assumes a monotonic or linear relationship between variables, and absence of the multicollinearity. This method produces coefficients that have a multiplicative relationship with the odds, facilitating the calculation of the influence of each independent variable on the dependent variable [57].

The importance of each independent variable was determined through permutation importance, with 10 iterations conducted to calculate the mean permutation importance scores and standard deviation for each variable, as shown in Table VI. Permutation importance assesses the significance of a variable on the prediction coefficients by repeatedly shuffling values of a single independent variable whereas other variables remain constant. This process, through iterations, reduces variability and randomness in the findings [58].

A synergistic effect of variable pairs was examined using conditional partial dependence plots, presented in Fig. 5. These plots reveal associative effects of features of interest on the dependent variable, including the direction of such effects. Unlike regular partial dependence plots, conditional partial dependence plots account for the influences of other independent variables [59].

This methodology stands out by simultaneously examining multiple project setup variables and their interactions, providing a more holistic view of the factors influencing code quality. This multi-dimensional approach is novel and fills a significant gap in existing research.

C. ANALYSIS - Impact of the code quality on academic performance

1) MANN-WHITNEY U TEST

To evaluate statistically significant causation between the CQ factors (0 and 1) and variables influencing student academic performance, identified as G and PCT, the Mann-Whitney U test was conducted, as shown in Table IX. The choice of the Mann-Whitney U test was due to the non-normal distribution of data and the ordinal scale of the dependent variables (G and PCT). This test is a non-parametric median test that compares differences in the dependent variable between two independent groups without requiring a specific data distribution, whether symmetric or asymmetric [60].

Before testing, two alternative hypotheses were designed.

Test 1 - the impact of the code quality on the grade.

- H0: The CQ does not have a statistically significant impact on student G.
- H1: The CQ has a statistically significant impact on student G.

Test 2 - the impact of the CQ on the PCT:

- H0: The code quality does not have a statistically significant impact on the PCT.
- H1: The code quality has a statistically significant impact on the PCT.

V. RESULTS

A. IMPACT OF PROJECT SETUP VARIABLES ON CQ

The analysis showed that SAU and TC have a statistically significant positive correlation to CQ, whereas PS has a statistically significant negative correlation to CQ. All other independent variables have not shown a statistically significant correlation to CQ, based on the significance level of 5%.

TABLE VI
CORRELATION BETWEEN PROJECT SETUP VARIABLES AND CQ

Variables	Tau	p
SAU	0.0543	0.0222
VCU	0.0748	0.0925
PS	-0.0954	0.0417
AY	0.0380	0.3019
TM	0.0632	0.1551
CM	-0.0405	0.3625
TC	0.1281	0.0105

All variables demonstrating a statistically significant correlation with the dependent variable exhibited a Variance Inflation Factor (VIF) lower than 5. The sole independent variables possessing a VIF exceeding 5 were VCU and AY. The absence of high collinearity among variables is one of the fundamental assumptions of the Binomial Logistic Regression model.

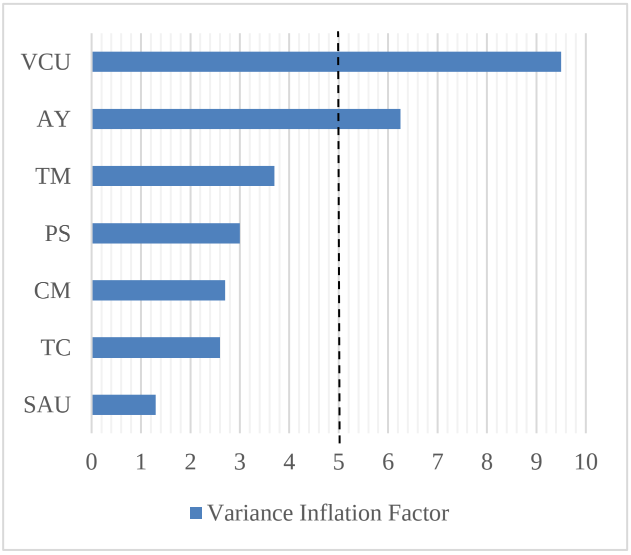

FIGURE 4. Variance Inflation Factor (VIF) – serves as an indicator of multicollinearity among predictors in a regression model, showing the degree to which the variance of an estimated regression coefficient is inflated due to correlations among the predictors.

The logistic regression analysis revealed a statistically significant causal relationship between SAU, PS, and TC with CQ. SAU emerged as the only input exhibiting a positive relationship with the dependent variable. Specifically, the utilization of static code analysis tools during development was associated with a 1.31-fold increase in the odds of producing high-quality code, as indicated by the odds ratio value. If students used .NET and Angular or only .NET, the odds of developing high-quality code are multiplied by 1.68. Conversely, PS is negatively related to the target variable. The odds ratio for PS indicated a roughly 44% decrease in the odds of developing high-quality code with each one-standard-deviation increase in the number of lines of code (PS). Results of Binomial Logistic Regression are presented in Table VII.

TABLE VII
BINOMINAL LOGISTIC REGRESSION RESULTS

Variables	Coefficient	Std. error	z value	Pr(>|z|)	OR
SAU	0.2679	0.1066	2.5117	0.0120	1.3072
PS	-0.5790	0.1308	4.4272	0.0000	0.5604
TM	0.1022	0.1138	0.8981	0.3691	1.1076
CM	-0.1087	0.1491	-0.7288	0.4661	0.8970
TC	0.5214	0.1455	-3.6453	0.0003	1.6844

To evaluate the importance of each variable, permutation importance was calculated across 10 iterations. This analysis demonstrated that all three variables, SAU, PS, and TC, held comparable levels of importance in the model's predictive accuracy, each exhibiting an extremely low standard error, presented in Table VIII.

TABLE VIII
CORRELATION BETWEEN INDEPENDENT VARIABLES AND CQ

Variables	Tau	p
SAU	0.1259	0.0000
PS	0.1375	0.0039
TC	0.1259	0.0000

. In the final phase of analysis, the synergistic effect of those three independent variables, SAU, PS, and TC, on the dependent variable (CQ) was examined using partial dependence plots. For this purpose, a new Binomial Logistic Regression model was fitted, this time utilizing non-transformed inputs to ensure interpretability. Among the interactions explored, a demonstrable synergistic effect was identified exclusively between the PS and TC variables and presented in Fig. 5. By combining a lower project size (number of lines of code) with technologies encoded in the range 120 – 130, t.e. the .NET and Angular or only .NET, the chances for high-quality code development are increasing

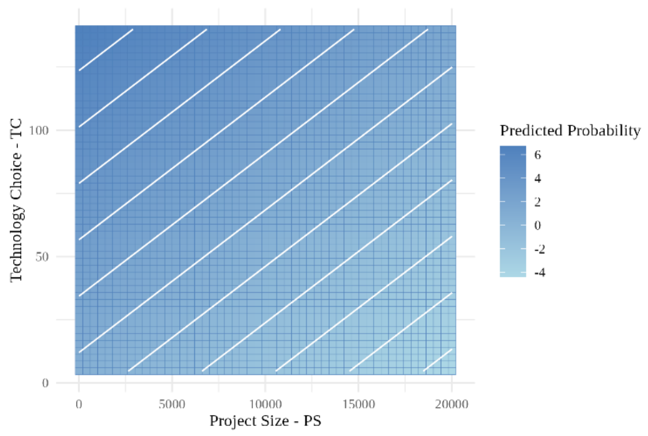

FIGURE 5. **Interaction effect – an indicator of the combined influence of Project Size (PS) and Technology Choice (TC) on Code Quality (CQ)**

B. RELATIONSHIP BETWEEN THE CQ AND STUDENT ACADEMIC PERFORMANCE

The Mann-Whitney U test was applied to explore disparities in grades between students who surpassed the CQ threshold and those who did not. Findings suggested that students with higher CQ exhibited marginally superior median exam G and reduced variance (Median = 9, Interquartile Range (IQR) = 2) relative to their counterparts with lower CQ (Median = 9, IQR = 3). The outcomes of the Mann-Whitney U tests revealed a statistically significant causal relationship between CQ and G. The effect size was further quantified through the calculation of Cliff's Delta, which stood at 0.3189, indicating a moderate effect magnitude on G in favor of students with higher CQ.

Additionally, the Mann-Whitney U test was employed to investigate variations between the CQ groups concerning the PCT. Results showed no difference in median and interquartile range between students with higher CQ and those with lower (Median = 2, IQR = 1), implying that the time taken to complete a project does not significantly influence the quality of the code produced.

These correlations are presented in Table IX.

TABLE IX
MANN-WHITNEY U TESTS RESULTS

Variables	W	p
CQ $\rightarrow$ G	26830	0.0029
CQ $\rightarrow$ PCT	29562	0.1971

VI. DISCUSSION AND LIMITATIONS

This section discusses the findings from the previous section, highlighting key factors that impact code quality in student projects and exploring their implications for software development education.

A. IMPACT OF PROJECT SETUP VARIABLES ON CQ

The results obtained from the analysis emphasize that certain project setup variables significantly influence the quality of the code generated by students. Notably, the use of static code analysis tools during project work is positively correlated with higher code quality, suggesting these tools are crucial in guiding students toward better coding practices. This finding underlines the notion that incorporating these tools into the educational process can markedly enhance the learning experience by offering instant feedback on potential irregularities and opportunities for source code enhancement.

Conversely, the observed variable Project Size (PS) shows a negative correlation with code quality, indicating that larger projects are more likely to encounter quality problems. This outcome could be attributed to the increased complexity and a higher likelihood of irregularities in larger codebases. It indicates the need for educators to carefully consider the scope of projects assigned to students, perhaps emphasizing incremental development and continuous testing to manage complexity and maintain high code quality.

The variable Technology Choice (TC) also emerged as a significant factor for generating high-quality code, with specific technologies leading to better outcomes. This finding points to the importance of selecting appropriate technologies that not only align with educational goals but also support best coding practices and project management. It is also noteworthy that some technologies might have a more challenging learning curve for students, which could affect their code quality.

Interestingly, variable, Collaboration Mode (CM) did not show a statistically significant impact on Code Quality (CQ) in this sample. This outcome suggests that the decision of whether a project is undertaken individually or in a team, may not be a critical factors when organizing a project. Teamwork often yields better results compared to individual tasks [19], [20]. Yet, it is not uncommon for only a small number of team members, or even a single participant, to be actively involved in team projects. This factor was not considered in this analysis and represents a foundation for further examination of this variable and its significance regarding code quality.

The Teaching Method (TM) variable also did not show a statistically significant impact on Code Quality (CQ) according to this analysis. In this context, it implies that the choice between conducting courses in-person or online does not affect the code quality that students produce in their projects. Although the online teaching environment necessitated by the COVID-19 pandemic has affected many aspects of education [24], [61], this study presents different findings, indicating that online instruction does not significantly impact the quality of code generated by students.

These observations do not diminish the possible value and importance of teamwork or the potential benefits of different teaching methods, which could affect other learning aspects not directly measured through code quality.

B. CORRELATION BETWEEN CQ AND ACADEMIC PERFORMANCE

Within the research findings, the correlation between Code Quality (CQ) and students' academic performance was analyzed. The analysis reveals a clear correlation between these variables, specifically showing that students with higher grades (G) generate source code of better quality. This correlation underscores the importance of code quality not just as a technical measure, but also as an educational outcome that reflects students' understanding and application of software engineering principles, and presents the possibility of using code quality analysis results to generate the grade a student achieves in the course. Furthermore, this correlation also indicates that teachers who have reviewed the projects possess the ability to recognize code quality and award corresponding grades

However, no significant correlation was found between Code Quality (CQ) and Project Completion Timeframe (PCT). This indicates that the time required to complete projects does not affect the quality of the generated code, which may imply that students who take longer are investing time in refining and improving their work to achieve higher quality. This finding suggests the need to investigate additional variables that could influence the timing of project submissions, such as the complexity of the project, the clarity of assignment instructions, and the presence of other academic obligations.

One of the most frequently mentioned limitations of static code analysis tools is the possibility of reports generating false positive or false negative results [1], [45]. This research did not consider the possibility of such outcomes, which could affect the CQ variable used for analysis to a certain extent.

Regarding the variable within the group describing students' academic performance, the G variable did not account for other factors that might influence the grade a student receives in the course, such as quizzes, tests, theoretical exams, etc.

The limitation regarding the Academic Year (AY) variable, which is supposed to describe the experience possessed by the student, excludes the possibility of other indicators of the student experience, such as knowledge and skills obtained outside of the university, as well as previously acquired experience and knowledge (for example, in earlier education). These factors were not included in the assessment of student experience in this research.

Another limitation of this research is that the sample on which the dataset was created was obtained only from courses across different programs at the Faculty of Technical Sciences, University of Novi Sad. Part of further research in the field certainly involves expanding the dataset with data from various technical universities in Serbia, as well as in other countries.

VII. CONCLUSION

The conducted research offers insights into factors affecting the code quality in student software projects and the relationship between code quality and academic performance. The positive impact of static code analysis tools on code quality highlights the importance of integrating these tools into computing education to support student learning and development. Additionally, findings on project size and technology choice provide practical guidelines for educators in creating effective and supportive learning environments.

Although the study did not find a significant impact of collaboration mode or teaching method on code quality, this suggests that from the perspective of code quality in student projects, the project setup in courses can be organized either as team-based or individual work, depending on the instructor's preference. Additionally, the results of this study indicate that from the standpoint of the code quality produced by students, the course curriculum can be effectively delivered both in-person and online.

The correlation between code quality and academic performance further emphasizes the need for educational strategies that focus not only on technical skills but also on the quality of work produced by students. By promoting an environment that prioritizes code quality, educators can better prepare students for the challenges of professional software development.

This research provides a detailed analysis of the factors influencing code quality in educational contexts. The findings offer valuable implications for improving teaching strategies and curriculum development in software engineering education. The unique contributions of this multi-dimensional approach are highlighted, clearly distinguishing this work from existing studies in the same area.

One possible direction for future research involves increasing the model's complexity (order of polynomials and regularization) to develop a predictive classification model, with the need to define and monitor additional metrics such as true positives, false positives, true negatives, false negatives, recall, and precision on unseen data.

This research contributes to the ongoing discussion on improving learning outcomes in computing education and lays the groundwork for future studies that could further explore additional factors and interventions that could further enhance both code quality and academic performance in computing disciplines. Building upon these findings, it is anticipated that this research will pave the way for the development of improved learning strategies in software engineering, ultimately fostering a new generation of software engineers who are well-equipped to navigate the complexities of contemporary software development.

ACKNOWLEDGMENT

This research has been supported by the Ministry of Science, Technological Development and Innovation (Contract No. 451-03-65/2024-03/200156) and the Faculty of Technical Sciences, University of Novi Sad through the project “Scientific and Artistic Research Work of Researchers in Teaching and Associate Positions at the Faculty of Technical Sciences, University of Novi Sad” (No. 01-3394/1).

REFERENCES

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

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

[3] 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.

[4] 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.

[5] 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.

[6] A. Katin, N. Taušan, V. Lenarduzzi, and V. Mandić, *Demystifying Sonar Tool Estimates in the Contexts of Familiar and Unfamiliar Software Projects: An Empirical Study with Junior Developers*. 2023. doi: 10.24867/IS-2023-T4.1-6_03541.

[7] M. L. Shooman, *Software Quality: Theory and Management*. Chapman & Hall, 1996.

[8] H. Keuning, B. Heeren, and J. Jeuring, *Code Quality Issues in Student Programs*. 2017, p. 115. doi: 10.1145/3059009.3059061.

[9] “ISO/IEC 25010:2011, Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.” International Organization for Standardization, 2011. [Online]. Available: https://www.iso.org/standard/35733.html

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

[20] 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. TOCE*, vol. 21, no. 3, pp. 1–22, 2021.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] “Bitbucket.” [Online]. Available: https://bitbucket.org

[27] 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.

[28] “GitHub.” [Online]. Available: https://github.com

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] D. Nikolić, S. Havzi, D. Dakić, T. Lolić, and D. Stefanović, "EVALUATION OF STRATEGIES OVER STATIC CODE ANALYSIS TOOLS," 2021.
[37] 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.
[38] 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.
[39] 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.
[40] D. Oguz and K. Oguz, "Perspectives on the Gap Between the Software Industry and the Software Engineering Education," *IEEE Access*, vol. 7, pp. 117527–117543, 2019, doi: 10.1109/ACCESS.2019.2936660.
[41] Cppcheck Development Team, "CppCheck." [Online]. Available: http://cppcheck.sourceforge.net
[42] The FindBugs Project, "FindBugs." [Online]. Available: http://findbugs.sourceforge.net
43] PMD Project, "PMD." [Online]. Available: https://pmd.github.io
[44] Splint Developers, "Splint." [Online]. Available: http://www.splint.org
[45] 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.
[46] 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.
[47] 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.
[48] V. Lenarduzzi, V. Mandić, A. Katin, and D. Taibi, *How long do Junior Developers take to Remove Technical Debt Items?* 2020. doi: 10.1145/3382494.3422169.
[49] V. Lenarduzzi, N. Saarimäki, and D. Taibi, "Some SonarQube issues have a significant but small effect on faults and changes. A large-scale empirical study," *J. Syst. Softw.*, vol. 170, p. 110750, Jul. 2020, doi: 10.1016/j.jss.2020.110750.
[50] M. Baldassarre, V. Lenarduzzi, S. Romano, and N. Saarimäki, "On the Diffuseness of Technical Debt Items and Accuracy of Remediation Time When Using SonarQube," *Inf. Softw. Technol.*, vol. 128, p. 106377, Jul. 2020, doi: 10.1016/j.infsof.2020.106377.
[51] "SonarQube Documentation." 2024. [Online]. Available: https://www.sonarqube.org/docs/
[52] 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.
[53] 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.
[54] 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.
[55] 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.
[56] J. Fox, *Applied Regression Analysis and Generalized Linear Models*, 3rd ed. Los Angeles: Sage Publications, 2016.
[57] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: With Applications in R*. New York: Springer, 2013.
[58] A. Chamma, D. A. Engemann, and B. Thirion, "Statistically Valid Variable Importance Assessment through Conditional Permutations," *ArXiv*, vol. abs/2309.07593, 2023, [Online]. Available: https://api.semanticscholar.org/CorpusID:261823259
[59] C. Molnar *et al.*, *Relating the Partial Dependence Plot and Permutation Feature Importance to the Data Generating Process*. 2023, p. 479. doi: 10.1007/978-3-031-44064-9_24.
[60] K. Rubarth, P. Sattler, H. Zimmermann, and F. Konietschke, "Estimation and Testing of Wilcoxon–Mann–Whitney Effects in Factorial Clustered Data Designs," *Symmetry*, vol. 14, p. 244, Jan. 2022, doi: 10.3390/sym14020244.
[61] K. Qin, X. Xie, Q. He, and G. Deng, "Early Warning of Student Performance With Integration of Subjective and Objective Elements," *IEEE Access*, vol. 11, pp. 72601–72617, 2023, doi: 10.1109/ACCESS.2023.3295580.

DANILO NIKOLIĆ (Graduate Student Member, IEEE) was born in Vršac, Serbia, in 1996. He received the B.S. and M.S. degrees in information systems engineering from the Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia, in 2019 and 2020, respectively, where he is currently pursuing a Ph.D. degree in information systems engineering. From 2019 to 2020, he was a Teaching Associate and since 2020, he has been a Teaching Assistant at the Department of Industrial Engineering and Engineering Management, Chair for Information and Communication Systems, Faculty of Technical Sciences, University of Novi Sad. His research interests include static code analysis and tools for static code analysis.

DARKO STEFANOVIĆ (Member, IEEE) was born in Novi Sad, Serbia, in 1972. He received a B.S. degree in mechanical engineering and a M.S. and Ph.D. degree in industrial engineering and engineering management from the Faculty of Technical Sciences, University of Novi Sad, Novi Sad, in 1999, 2005, and 2012, respectively. From 2001 to 2012, he was a Teaching Assistant, and from 2012 to 2017, he was an Assistant Professor, and Associate Professor from 2017 to 2022 at the Department of Industrial Engineering and Engineering Management, Chair for Information and Communication Systems, Faculty of Technical Sciences, University of Novi Sad, where he has been an Full Professor, since 2022. He is currently the Vice-Dean for Science and International Cooperation, at the Faculty of Technical Sciences, at the University of Novi Sad, and the Head of the Chair of Information and Communication Systems. He has published in several international information systems journals. His research interests include ERP systems, e-learning systems, e-government systems, data mining, static code analysis and business process mining in production planning

MIROSLAV NIKOLIĆ (Graduate Student Member, IEEE) was born in 1991. He received B.A. in Research Sociology and M.S. in Applied Data Science and Artificial Intelligence from University of Belgrade - Serbia and Open Institute of Technology - Malta, respectively. He wrote his theses in the realm of applied statistics and machine learning.

He is currently enrolled as a PhD student at Faculty of Organizational Sciences, University of Belgrade, focussing on numerical optimization advancements in foundation models and transformer architectures.

DUŠANKA DAKIĆ. (Member, IEEE) was born in Novi Sad, Serbia in 1993. She received her B.S. degree in Engineering Management in 2016, her M.S. degree in Information Systems Engineering in 2017, and her Ph.D. degree in engineering management in 2023 from the Faculty of Technical Sciences at the University of Novi Sad, Serbia.

From 2016 to 2024, she was a Research Assistant, and since 2024, she has been an Associate Professor with the Engineering Management Department at the Faculty of Technical Sciences, University of Novi Sad, Serbia. She is the author of 20 conference papers and 5 journal articles. Her research interests include Data Quality, Code Quality, Static Code Analysis, Business Process Management, Process Mining, and Business Information Systems.

MIROSLAV STEFANOVIĆ (Member, IEEE) was born in Loznica, Serbia, in 1975. He received a B.S. degree in information management and an M.S. and Ph.D. degree in information systems engineering from the Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia, in 2014 and 2023, respectively, From 2015 to 2016, he was a Teaching Associate, and from 2016 to 2023, he was a Teaching Assistant, and since 2023 he has been an Assistant Professor at the Department of Industrial Engineering and Engineering Management, Chair for Information and Communication Systems, Faculty of Technical Sciences, University of Novi Sad. His research interests other than code reviewing and static code analysis, include blockchain technologies, especially the implementation of blockchain technology in fields other than cryptocurrency, mainly e-government and land administration systems.

SARA KOPRIVICA. (Graduate Student Member, IEEE) born in 1996 in Novi Sad, Serbia, holds a bachelor's degree with honors and an MSc degree in information technology from the University of Novi Sad. Currently pursuing her doctoral studies in the same field, Sara serves as an assistant at the Department of Information and Communication Systems at the Faculty of Technical Sciences, University of Novi Sad. In addition to her academic pursuits, Sara has gained research experience through her engagement as a researcher at the National Research Council in Palermo, Italy. Her professional journey extends to consultancy roles in various projects concerning information technology and electronic governance. Her research interests include information technology, information systems, static code analysis, electronic governance, technology acceptance models, and automation of the literature review. Sara is the author of more than 20 research papers.