




# Plagiarism Detection and its Effect on the Learning Outcomes

1<sup>st</sup> Jonnathan Berrezueta-Guzman   
Technical University of Munich  
s.berrezueta@tum.de

2<sup>nd</sup> Markus Paulsen   
Technical University of Munich  
markus.paulsen@tum.de

3<sup>rd</sup> Stephan Krusche   
Technical University of Munich  
krusche@tum.de

**Abstract**—The consistent effort and autonomous problem-solving required in programming instruction presents unique educational challenges, with plagiarism being a significant factor negatively correlated with independent programming performance. The prevalence of digital communication and information exchange further complicates this issue.

This research paper aims to investigate and discern patterns of plagiarism among first-year computer science undergraduates. To investigate the research questions, we conducted three controlled experiments. The first experiment demonstrated that students were more likely to commit plagiarism when provided with the opportunity to do so. The second experiment corroborated this, showing a tendency for plagiarism among students who had not previously been exposed to any plagiarism mitigation strategies. The third experiment revealed that early awareness of plagiarism mitigation strategies leads to enhanced student performance, indicating that awareness of these measures acts as a stimulant for autonomous learning.

**Index Terms**—programming education, interactive learning, online training and education, software engineering education for novices, vision for education in the future

## I. INTRODUCTION

Plagiarism poses a substantial challenge in introductory programming courses, presenting both immediate and long-term consequences for students [1]. Extensive research has established a strong correlation between plagiarism and academic failure, with students who engage in plagiarism consistently achieving the poorest performance in individual examinations [2]. Moreover, the implications of plagiarism extend beyond the academic realm, impacting students' professional lives and imposing limitations on their subsequent performance [3], [4].

The task of plagiarism detection in assignments within large-scale courses presents a formidable challenge. This complexity arises from the fact that the required effort for plagiarism detection is directly proportional to the number of students, while the number of reviewers available for the task usually remains constant [5]. Without the aid of specialized tools, determining instances of plagiarism, as well as identifying the originators/creators (providers) and recipients (takers) of plagiarized content, becomes a non-trivial problem within the domain of computer science education [6].

This paper makes a significant contribution by conducting three rigorous experiments to validate key hypotheses. Firstly, the findings confirm that the presence of an opportunity to do so significantly increases the likelihood of students engaging in plagiarism. Secondly, the results demonstrate that students

who lack prior exposure to plagiarism control measures are more prone to committing plagiarism. Lastly, the study reveals that early communication of plagiarism control measures at the beginning of the course serves as a motivational factor for students, leading to improved academic performance.

Notably, the outcomes of the third experiment hold substantial implications. The data unequivocally indicate that students who are aware of and adhere to plagiarism control measures achieve their intended learning outcomes more effectively (achieving higher grades). This accomplishment in turn helps them to avoid complications in subsequent subjects throughout their academic journey, safeguarding the quality of their education. Moreover, these positive outcomes extend beyond the realm of academia, benefiting students in their future professional opportunities.

This paper systematically presents its content in the following way: Section II is dedicated to reviewing and summarizing relevant existing related work, along with their respective conclusions and limitations. Section III explains the methodology adopted in this research. It outlines the organizational aspects of the conducted courses and describes the specific tool employed for plagiarism control. Section IV presents a meticulous overview of the experimental design, data collection procedures, and the subsequent results derived from the experiments.

In Section V, the paper offers a discussion to critically analyze and interpret the obtained results. This section serves to emphasize the significance of the findings, considering their implications and potential ramifications. Finally, Section VI concludes the research by summarizing the key insights and conclusions drawn from the study. It also presents potential avenues for future research, highlighting areas that warrant further investigation and suggesting potential directions for improving plagiarism control.

## II. RELATED WORK

According to Jenkins and his colleagues, plagiarism happens mostly when the student encounters the opportunity [7]. Additionally, some researchers experimented and concluded that plagiarism is related to high levels of failure in academic courses and especially those involving programming exercises [8], [9]. Therefore, including a plagiarism control system for the source code of programming assignments has motivated

students to work autonomously and to improve their academic performance [4], [10].

Kaya and Özel extended the Moodle system with the GNU Compiler Collection (GCC)<sup>1</sup> and the Measure of Software Similarity (MOSS)<sup>2</sup> source code plagiarism detection tool to decrease the effort for the assessment of programming assignments and prevent students from plagiarism in the Data Structure course [13]. The experiment was conducted with two courses of the same subject, and the same professor but in different years.

The results indicated that the first group (which was not aware of the plagiarism check) was the one with lower performance (29.8 % of students failed) in comparison to the second group (which was aware of the plagiarism check) where 21.6 % of students failed (8.2 % lower failure rate). As it was expected, the first group was the one with a higher amount of plagiarism cases, taking into account a threshold of 50 % of similarity. However, the experiment had only 3 programming assignments and also included 1 midterm and 1 final exam.

Pawelczak used a tool for *tokenizing* and averages certain characteristics of the source code [14]. He analyzed 5 years of data from a course on the introduction of programming in C language with 7 programming exercise assignments. The typical threshold used was 80 % in this study due to the average length of the exercises.

The results established that since the implementation of the tool, the performance of the students in examinations is about 8.6 % better. They also find that about 84 % of the students who failed plagiarized in the programming assignments. Additionally, the tool detected about 92 % of students who abandoned the course were involved in committing plagiarism.

Pawelczak also implemented an experiment where he compared two introductory programming courses on their assignments and their learning outcomes [15]. He checked the students' submissions (for 7 assignments in total) for plagiarism directly after the submission deadline while for the second one, we conducted the plagiarism check after the semester ended.

The results indicate that the average percentage of students plagiarizing in the first group is 11.1 %, while in the second group is 35.1 % (24 % higher than the first group). Additionally, the second group was the one with lower performance in the final examination. However, the students in this group presented better skills in fundamental coding. Pawelczak believes this is due to the students being forced to edit their implementations (adding loops, outsourcing code into functions, etc.) in order not to be caught by the plagiarism detection tool.

On the other hand, Halak and El-Hajjar presented two techniques to prevent plagiarism [16]. The first one is assigning a

unique exercise per student while the second one is conducting individual presentations. The first technique suggested a considerable reduction in plagiarism detection in different classes, as there was a reduction of inter-report similarity of 13 %. The second technique suggested that its application motivated the students to avoid plagiarism. However, Halak and El-Hajjar didn't mention the necessary effort of the instructors of conducting these techniques.

Finally, Moss and his colleagues conducted a systematic review of 83 empirical papers to clarify the psychological causes of plagiarism [17]. The finding of this study established that the possible causes can be the emphasis on competition and success rather than development and cooperation coupled with impaired resilience, limited confidence, impulsive tendencies, and biased cognitions.

These studies and their results collectively contribute to the understanding of plagiarism in programming courses and provide insights into effective approaches for plagiarism detection, prevention, and fostering academic integrity among students.

### III. METHODOLOGY

This research study aims to address three primary research questions:

- **RQ1:** Do students exhibit a higher propensity to engage in plagiarism when presented with an opportunity?
- **RQ2:** Does prior experience with plagiarism checks enhance students' motivation to work autonomously, compared to those without such experience?
- **RQ3:** Does the announcement of plagiarism checks at the beginning of the course yield beneficial outcomes for students?

To address these questions, we devised and executed three experiments spanning three semesters within the Information Engineering bachelor's degree program: winter semester 2021-2022 (WS21/22), summer semester 2022 (SS22), and winter semester 2022-2023 (WS22/23).

During the winter semesters, we conducted an **introductory programming course (InProg)** targeted at first-semester students, facilitating their progression from beginner to intermediate-advanced programming skills. In the summer semester, we offered an **Introduction to Software Engineering (ISE)** course, intended for second-semester students to develop advanced to professional-level software engineering and programming skills.

While it is not mandatory to have completed InProg to enroll in ISE, students are expected to leverage their acquired knowledge from InProg to successfully tackle the proposed exercises and the final exam.

These courses employ interactive learning, which aims to minimize the temporal gap between theoretical learning and practical application by students [18]–[21].

<sup>1</sup>It is a free and open-source compiler system developed by the GNU Project. It is used to compile various programming languages such as C, C++, Objective-C, Fortran, Ada, and others [11].

<sup>2</sup>It is an automatic system for determining the similarity of programs. To date, the main application of Moss has been focused on detecting plagiarism in programming classes [12].

## A. Course Organization

**InProg.** The introductory programming course takes place over 12 weeks in which students not only take the lecture but also work on exercises that prepare them for the final exam.

**Quiz exercises.** At the beginning of each lecture, students are presented with a set of 3-5 questions associated with the topics covered in the previous lecture. They are given 5 minutes to solve these quiz exercises. When the time is up, the instructor proceeds to explain the solution and then addresses any doubts or objections raised by the students afterwards.

**Tutor exercises.** These exercises consist of two assignments per week and primarily focus on programming tasks of easy to medium difficulty. Students collaboratively work on these exercises during tutoring sessions, fostering the development of soft skills such as effective communication and teamwork. The collaborative nature of these exercises prepares students for independent problem-solving in subsequent homework exercises.

**Homework exercises.** These assignments comprise two weekly programming exercises of medium to hard difficulty. The submissions of these exercises contribute to the practical part (Prog) of the InProg course's final grade and aid in preparing students for the final exam, which determines the theoretical part (In) grade of the course.

**Presentations.** Homework exercises and tutorial exercises are presented by students each week during the tutorial sessions. This is a suitable environment to discuss the solutions and resolve any remaining questions. This is an additional step to check if students can understand and present their own solution.

**ISE.** The course Introduction to Software Engineering follows a similar methodology to InProg, but with the inclusion of other exercise types, such as text exercises and modeling exercises, in addition to programming tasks. Students take it in their second semester, as it builds on the InProg.

## B. Plagiarism Detection Process and Tool

Artemis is an open-source learning and research platform<sup>3</sup>. It is designed to facilitate the creation and automated assessment of programming, modeling, text, and file uploading exercises [22]. Students participate in programming exercises by committing and pushing code in a version control repository. They receive feedback from automatically executed tests and static code analysis in a continuous integration environment. Artemis incorporates a semi-automatic system for detecting similarity among submissions of the same exercise [23].

In order to achieve this, Artemis integrates JPlag, a plagiarism detection service equipped with a Greedy-String-Tiling algorithm [24]. JPlag is capable of identifying pairs of similar programs within a given set of programs and supports multiple programming languages, including C, C++, Java, and more. It compares the abstract syntax tree of two programs. This way, the comparison hits even if students copy the code and make simple changes to variable names or the order of statements.

While other tools may employ different detection methodologies, such as Neural Networks [25], JPlag has proven to be highly efficient in terms of analysis time and accuracy when identifying instances of plagiarism among submissions.

When students upload their solutions through Artemis, instructors have the option to manually initiate a similarity analysis run by setting a threshold based on the exercise's length. For instance, the threshold can be determined by the average number of lines of code required to complete a programming-based exercise, the length of words in a text-based exercise, or the components used in a model-based exercise.

Figure 1 illustrates the sequential steps involved in the automatic checking, manual analysis, student notification, student statement, and verdict of a plagiarism case, carried out throughout the utilization of the Artemis platform. These steps contain the following actions:

**1. Automatic check:** Once the exercise instructors activate the automatic plagiarism checker in the plagiarism detection screen in Artemis by clicking on the "Detect Plagiarism" Button, all student submissions are compared against each other using JPlag's greedy string tiling algorithm in order to obtain a measure of the similarity of a submission pair (in percent) and a plagiarism report describing the similarities found. Artemis now displays to the exercise instructors every submission pair above the aforementioned threshold and an overall similarity distribution.

Overall, this step preprocesses and discards a significant number of submission pairs that are not suspected to be involved in plagiarism (indicated by a similarity measure below the threshold). Therefore, the exercise instructors can focus exclusively on the cases which are suspected to be plagiarism cases.

**2. Manual analysis:** The exercise instructors inspect the plagiarism report, which displays the two similar code files side by side. The part of the code that was not included in the template and overlaps is highlighted in blue. Subsequently, the exercise instructors decide whether the similarities found are indeed indicative of plagiarism.

Therefore, they try to spot typical plagiarism patterns and other recognizable indications like removing/adding white spaces, renaming variables, identical variable naming strategies, changing between similar types of control statements, identical comments, identical text patterns, identical mistakes, and identical elements within a UML diagram. In case there is sufficient reasonable evidence of plagiarism, the exercise instructors will mark the submission pair as a "plagiarism case". Otherwise, they will mark the submission pair as a "no plagiarism case".

This step is necessary as automatic checking is not yet able to accurately detect all cases and would produce too many false positives. Therefore, the exercise instructors manually review all cases with high similarities and accept or deny each case.

**3. Student notification:** If the exercise instructors (after a second screening) decide a particular pair of submissions

<sup>3</sup><https://github.com/ls1intum/Artemis>

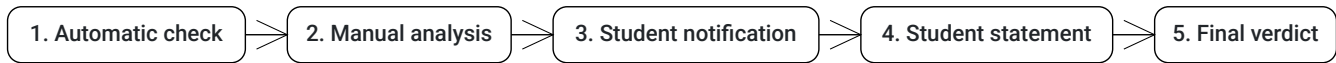


Fig. 1. Procedure of the plagiarism check: The instructors evaluate the results of the automatic check for plausibility and notify the identified students, who can submit a statement on their case. Depending on the severity of the plagiarism and the student’s statement, the instructors decide on the final verdict.

to be a case of plagiarism, they will issue a notification email in Artemis to both involved students. The email will inform them of the suspected plagiarism and the consequences they can expect (e.g., failing the course and only getting one more chance), and the opportunity to submit comments within the following seven days. In either case, both students are suspected of plagiarism, as there is usually no way to clearly identify the sender and the participant of a deception before contacting them. This step is required to inform the student of the suspected plagiarism, which may or may not subsequently result in a plagiarism conviction.

**4. Student statement:** The student responds to the notification by using the link embedded in the email. Using this link, the student can view the comparison between the own work and the anonymized work of the other involved student. The student explains in detail why the work is supposed to be original and therefore the suspicion is unfounded.

The student must demonstrate that the assumptions made by the exercise instructors regarding the plagiarism patterns detected were incorrect or that the accuracy of the detection was too low. Likewise, they have the opportunity to admit their misconduct and facilitate further proceedings. This step allows the student to justify the submission against suspicion of plagiarism, which may be warranted in the case of a false positive.

**5. Final verdict:** The exercise instructors meticulously assess and evaluate whether the justifications made by the student are steadfast. This involves analyzing whether the students’ justification demonstrate that they fully mastered the concepts taught in the lecture and exercises, and whether there are any unresolved gaps or problems in the reasoning of the exercise instructors.

Finally, the exercise instructors decide on a verdict of either “plagiarism” (in case the justifications were not steadfast), resulting in a failing grade, or “no plagiarism” (in case the justifications were steadfast), which has no immediate consequences, except that students are now generally more careful with their submissions. A third alternative would be to deduct points in the respective exercise and to issue a warning.

In this last step, the exercise instructors pronounce their final verdict for the two students involved, whereupon the appropriate consequences are announced. This final verdict is documented in Artemis together with the plagiarism report in case the data is needed later, e.g. if the student complains to the examination board or sues the universities in court.

The identification of the *taker* (the student who engaged in copying) and the *giver* (the student who facilitated the copying) can sometimes be ascertained through the discussions

conducted with the involved students in a plagiarism case. The determination of their roles relies partially on the admissions provided by the students who acknowledge their involvement.

Verification is feasible due to the availability of precise data in Artemis, including the specific number and timing of commits made by the students. Typically, the giver is characterized as the student who initiates and completes the task first, while also demonstrating a higher frequency of commits. Conversely, the taker often exhibits contrasting behavior, typically commencing the task later and generating fewer commits. By considering these observed patterns and leveraging the data provided by Artemis, instructors can discern the roles of the taker and giver in the plagiarism case.

#### IV. EXPERIMENT AND RESULTS

A total of three experiments were conducted over three semesters. The organization of these experiments, along with their respective pre-conditions, student groups, and the corresponding courses involved, are presented in Table I.

TABLE I  
ORGANIZATION OF THE EXPERIMENTS. AWARENESS MEANS WHETHER STUDENTS HAVE BEEN INFORMED ABOUT PLAGIARISM CONTROL.

Experiment	Students’ groups	Awareness	Course
WS21/22	HN01 (70 students)	Since the middle of the course	InProg
SS22	HN01 (44 students) GA01 (1621 students)	Since the beginning of the course	ISE
WS22/23	HN02 (61 students)	Since the beginning of the course	InProg

##### A. First Experiment - WS21/22

The objective of this experiment was to investigate whether students (HN01 group) in the InProg course engaged in plagiarism when they were not initially informed about the implementation of plagiarism checks. Additionally, we focused to assess their behavior after the announcement of the plagiarism checks.

In order to achieve this, we chose not to inform the students about the plagiarism control at the beginning of the semester but rather in the middle of the course period, specifically in week 8. During that week, we provided detailed information on how the checks were conducted (as described in the previous section) and the penalties.

Figure 2 illustrates that the plagiarism control was initiated in exercise H03E01 (where the x-axis starts), as the initial two weeks primarily involved simple exercises with minimal lines of code (conducting a plagiarism analysis on these exercises

would have resulted in an unnecessarily high number of plagiarism accusations).

We observed an average of 20 instances of similarity exceeding the predetermined threshold per exercise during the first 8 weeks (the number of possible cases of similarity likely depends on the difficulty level of each exercise). However, in week 8, during the lecture, we explicitly announced to the students that plagiarism checks would be conducted for assignments starting from week 9.

As a result, we subsequently observed a significant decrease of 84 % in the number of submissions with similarity above the threshold for homework H09E01. Despite the prior notification during the lecture, we had to issue plagiarism notifications to 4 students, indicating their involvement in plagiarism (First notification).

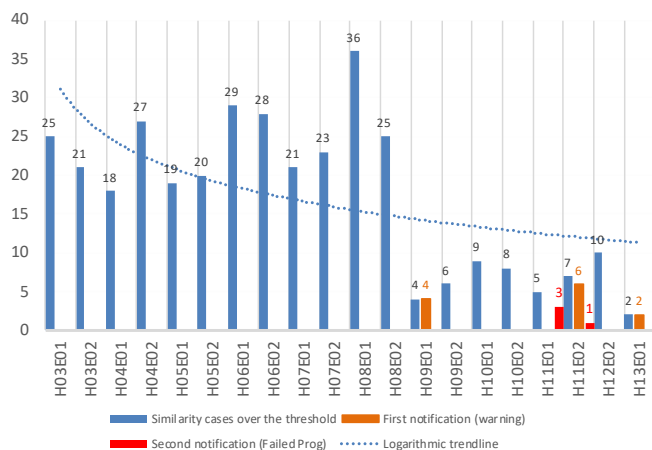


Fig. 2. Incidents of code similarity identified through the use of Artemis, in conjunction with notifications with potential instances of plagiarism throughout the course assignments.

Additionally, it is worth noting that certain submissions for homework H09E02, H10E01, and H10E02 exhibited a similarity level surpassing the predefined threshold. However, upon conducting a comprehensive review by the instructors, it was concluded that no instances of plagiarism were present in these submissions (false-positive cases). Consequently, the students involved were not notified of any plagiarism concerns.

In week 11, during the assessment of assignments H11E01 and H11E02, the plagiarism control was conducted, leading to the identification of 10 new cases of clear plagiarism involving students. Out of these 10 students, 4 were notified for the second time, resulting in immediate course failure. Additionally, 6 students received their first notification regarding plagiarized submissions. Overall, there were 8 students who received a first notification, while 4 students failed the course upon receiving the second notification.

Through an analysis of the time invested and commits made in each exercise, Figure 3 reveals that assignment H09E01, which included the initial 4 plagiarism cases, required the highest amount of time and had the most pushed commits by students. The instructors conclude that this particular exercise

was adequately complex, presented a specific context, and demanded students to propose their solutions. Furthermore, the notification of the plagiarism control served as an alert to refrain from copying, resulting in increased time investment and commits compared to the other exercises

Similar patterns were observed in exercises H09E02 and H13E01, where a substantial number of hours were invested and only a few instances of similarity were found. However, in the case of H13E01, these instances were confirmed as instances of complete plagiarism. Hence, it can be inferred that the occurrence of plagiarism in these exercises can be attributed to their complexity. The students were confronted with challenging exercises that pressured them to utilize plagiarism in order to resolve them.

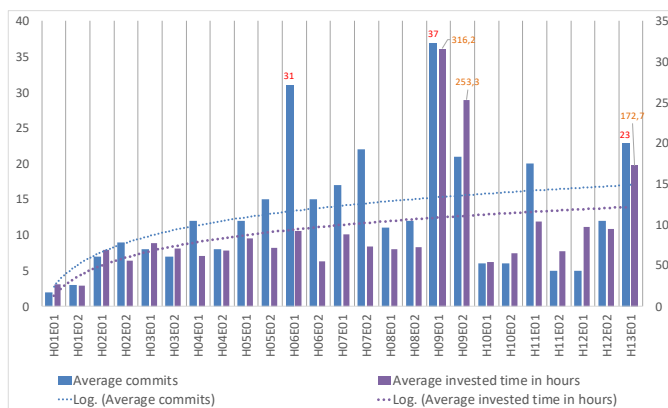


Fig. 3. Quantitative representation of the average commit frequency and the quantified time investment required to solve course exercises, as captured from the Artemis learning platform data.

Out of the 12 students who received at least one notification for a plagiarism case, 8 exhibited characteristics indicative of being takers. These students displayed a lower number of committed activities and invested less time in solving the exercises compared to their peers who were also involved in plagiarism cases.

Figure 4 illustrates that exercises H09E01 and H13E01 had the lowest levels of participation. With the exception of exercise H06E01, which had a high level of complexity (as indicated by the number of attempts in Figure 3), the complexities of the other exercises were generally at a medium level.

It was evident that, following the plagiarism control notification, students either strived to work independently or gave up. This finding confirmed our initial expectation, that the takers would be inclined to avoid making any effort, while the givers preferred not to share their solutions. Without counting with the tendency of quite this kind of courses.

At the conclusion of the course, a final exam was administered, with the programming component accounting for 70 % of the total grade. The analysis of the results revealed that the 4 students who received notifications for committing plagiarism and subsequently failed the practical part of the course also failed the final exam. Similarly, the 8 students



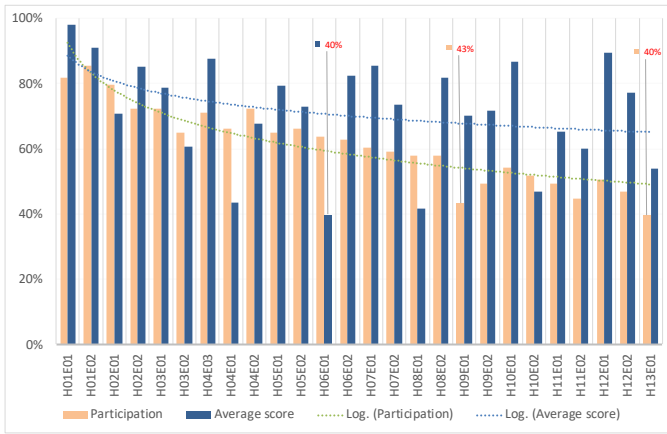


Fig. 4. Quantitative representation of student participation, alongside corresponding average grade distributions, throughout the duration of the course.

who exhibited characteristics consistent with being takers also failed the exam.

Hence, we can infer that these 8 students were indeed takers. This outcome validates the efficacy of our plagiarism analysis in the exercises, as it aligns with our initial expectations.

**Finding 1:** In the absence of plagiarism control, students are more likely to engage in plagiarism when presented with the opportunity.

Furthermore, it suggests that students who commit plagiarism, referred to as takers, generally exhibit lower performance in autonomous examinations. In contrast, students who facilitate plagiarism, referred to as givers, tend to demonstrate better performance.

**Finding 2:** Following a plagiarism control notification, students either strive to work independently or simply give up.

### B. Second Experiment - SS22

This experiment involved the comparative analysis of plagiarism behavior between two distinct groups of students within the same subject but in different locations. The first group consisted of students from Experiment 1, referred to as the HN01 group, while the second group comprised students who had not previously undergone a plagiarism control process, denoted as the GA01 group.

In this experiment, both groups were immediately informed about the implementation of a plagiarism check for all assignments at the beginning of the SS22 semester. The subject taught during this period was ISE, encompassing programming tasks, modeling exercises, and text-based assignments.

The average grade between the HN01 and GA01 groups exhibited minimal imbalance. The HN01 group achieved an average grade of 77.4 %, whereas the GA01 group obtained an average percentage of 78.3 %, resulting in a difference of less than 1 %. However, a notable divergence was observed

in the participation rates of the two groups. The HN01 group experienced a reduction in participation of 22.6 % (as shown in Figure 5), while the GA01 group exhibited a more significant decrease of 40.7 %, nearly double the reduction (as shown in Figure 6).

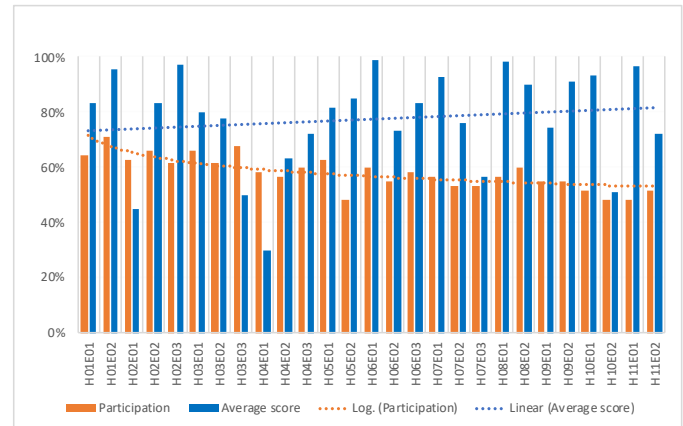


Fig. 5. Quantitative representation of student participation, alongside corresponding average grade distributions, of the HN01 group throughout the duration of the ISE course.

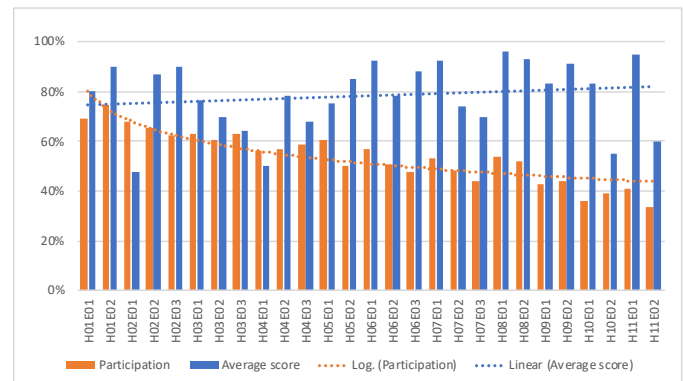


Fig. 6. Quantitative representation of student participation, alongside corresponding average grade distributions, of GA01 group throughout the duration of the ISE course.

The occurrence of detected plagiarism cases exhibited a substantial disparity between the HN01 and GA01 groups. In the HN01 group, only 3 exercises indicated a similarity level surpassing the established threshold, as shown in Figure 7. In contrast, the GA01 group indicated 15 exercises with a similarity level above the threshold, which suggests a quadruple increase compared to the HN01 group, as shown in Figure 8.

The distribution of plagiarism cases was more evenly spread across the exercises in the GA01 group. In the HN01 group, a total of 12 students were notified, out of which 2 students received a second notification and subsequently failed the course. On the other hand, in the GA01 group, a larger number of students, specifically 86 students, were notified of plagiarism cases, and among them, 22 students failed the course.

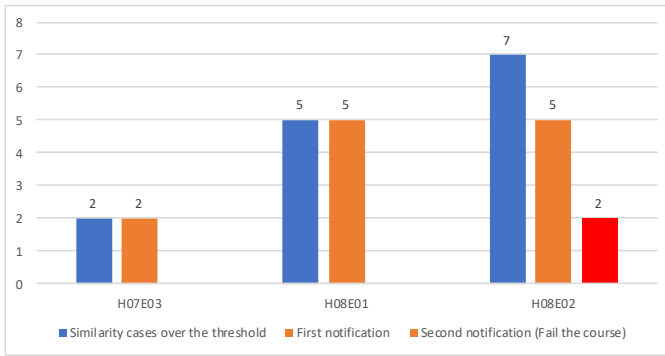


Fig. 7. Quantitative representation of the plagiarism cases, alongside corresponding notifications of HN01 group throughout the duration of the course.

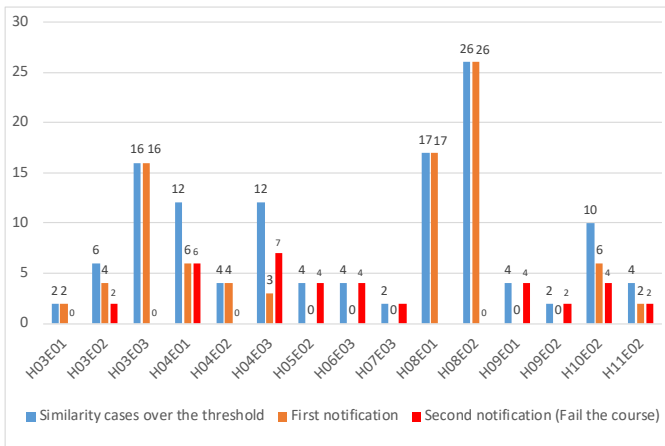


Fig. 8. Quantitative representation of the plagiarism cases, alongside corresponding notifications of GA01 group throughout the duration of the course.

**Finding 3:** Students without previous plagiarism control exposure tend to plagiarize more than those with plagiarism control experience. Awareness of plagiarism control also increases submission caution.

### C. Third Experiment - WS22/23

The third experiment, conducted in the WS22/23 semester, followed a similar approach to the first experiment (WS21/22), but with some modifications. In this experiment, all students (HN02 group) in the InProg course were informed about the presence of plagiarism control right from the beginning of the course. The student cohort consisted of 61 individuals, with 11 of them repeating the subject. Among these 11 students, 4 had previously failed the course in the WS21/22 semester by a plagiarism verdict.

In line with the findings from the previous experiment, the determination of the similarity threshold in this experiment was based on the number of lines of code. Additionally, exercises that tended to have a limited number of possible solutions were excluded from the plagiarism control analysis.

Furthermore, the number of assignments in this experiment was reduced to 22, with students receiving 2 weekly assign-

ments throughout the course duration (due to the semester planning, week 12 had no assignments).

Figure 9 shows a notable decline in the number of similarity checks, reaching zero starting from week three. In contrast, to experiment 1 with the HN01 group (Figure 2), there is no provision for a first notification in this experiment, and students fail the course directly upon confirmation of a plagiarism case.

Consequently, assignments (from week 4 onwards) exhibit no instances of similarity surpassing the established threshold. This new condition has led to a significant reduction in the instructor's effort required for reviewing similarity cases.

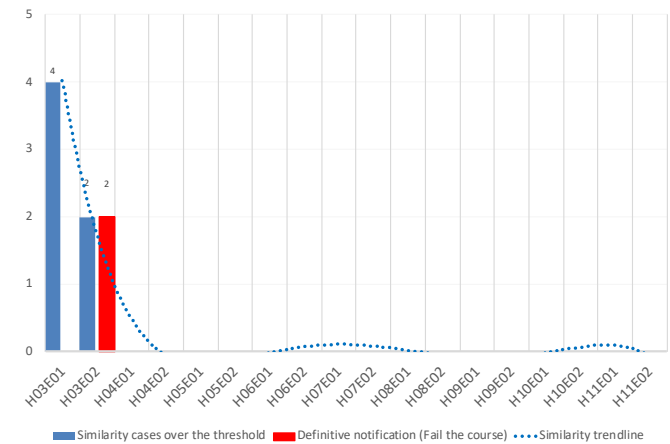


Fig. 9. Quantitative representation of the plagiarism cases, alongside corresponding notifications of HN02 group throughout the duration of the course.

Upon analyzing Figure 10, it is evident that the distribution of grades in the practical component of the course (Prog) exhibits a higher level of performance, as indicated by the average score of 1.7. This performance surpasses that of the HN01 group in the first experiment, as depicted in Figure 11, where the average score was 2.7. The high number of 5.0 students grades, is among the students who failed the course because of insufficient grades and also those students who quit the course.

The implementation of plagiarism control measures, coupled with an early announcement of its enforcement at the beginning of a course, not only serves to mitigate instances of plagiarism but also facilitates enhanced comprehension of programming concepts among students. This improvement in understanding is subsequently reflected in the grades achieved by the students.

Upon comparing the performance of the HN02 group, which experienced the early announcement and implementation of plagiarism control, with the HN01 group, it is evident that the former demonstrate superior performance in the final exam (In). The HN02 group achieved an average score of 2.7 (see Figure 12), whereas the HN01 group obtained an average score of 3.7 (see Figure 13).

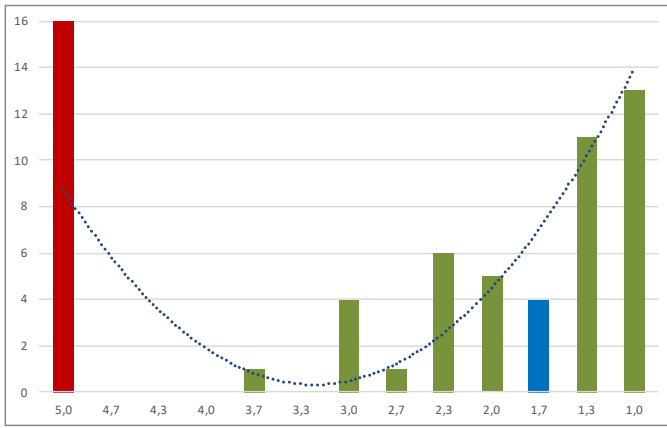


Fig. 10. Graphical representation of grade distribution for the practical programming course (Prog) during the winter semester WS22/23. The average grade of the course is highlighted in blue.

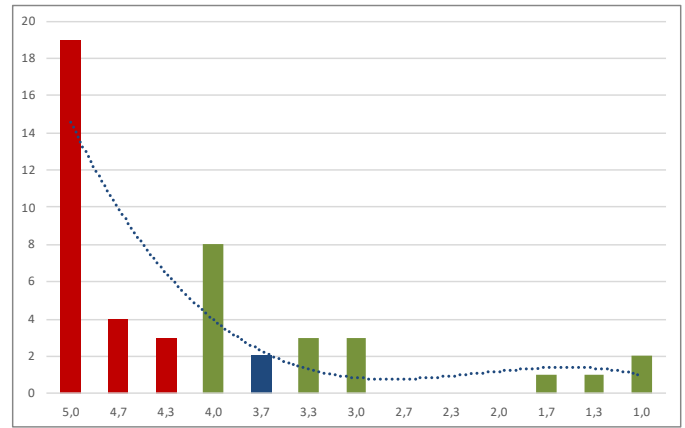


Fig. 13. Graphical representation of final examination grades (In) for the winter semester WS21/22. The average grade of the course is highlighted in blue.

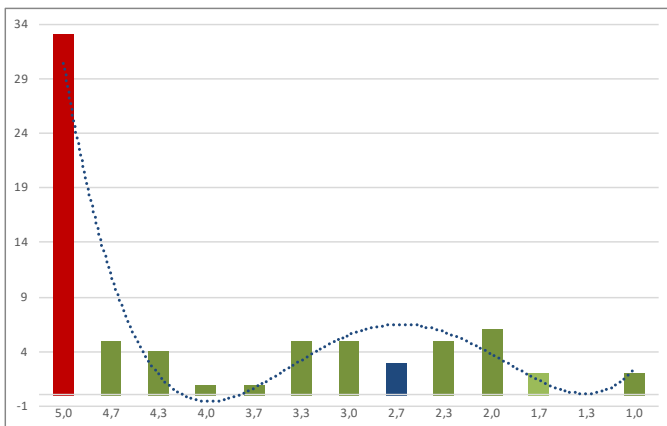


Fig. 11. Graphical representation of grade distribution for the practical programming course (Prog) during the winter semester WS21/22. The average grade of the course is highlighted in blue.

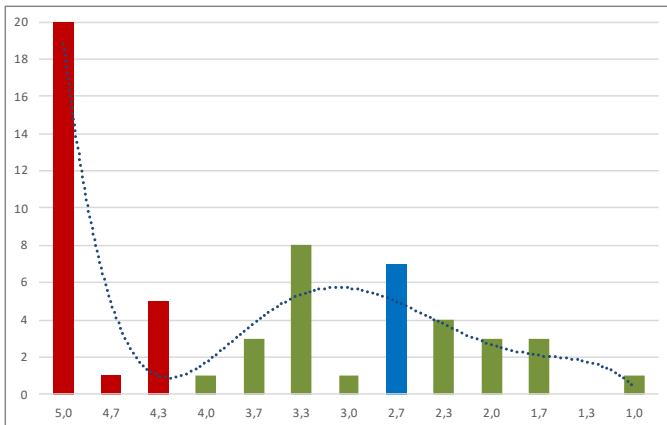


Fig. 12. Graphical representation of final examination grades (In) for the winter semester WS22/23. The average grade of the course is highlighted in blue.

**Finding 4:** Early disclosure of plagiarism control within the course significantly fostered student engagement in autonomous work.

This finding was validated by enhanced performance in individual exams. The improvement underscores the effectiveness of plagiarism control awareness in boosting programming skill acquisition. At this point, it is important to highlight that these three findings have the potential for broad applicability across various courses within the domain of software engineering and programming education.

## V. DISCUSSION

The research questions raised in this article have been addressed through a series of three experiments.

**1. Opportunity to commit plagiarism.** The findings of the first experiment shed light on some intriguing patterns of student behavior. It appears that students exhibit an elevated propensity to engage in plagiarism when they perceive an opportunity to do so. This inclination to engage in such practices is a notable observation and further emphasizes the need for vigilant plagiarism control measures in academic environments.

However, when the plagiarism control was announced and communicated clearly, the dynamics shifted noticeably. There was a significant reduction in the number of suspected cases of plagiarism by as much as 72 %. This statistic shows the deterrent effect that communication about plagiarism control measures can have on students' predisposition to plagiarize.

**2. No experience with plagiarism control.** The results obtained in the second experiment shed significant light on the effects of prior exposure to plagiarism control measures on student behavior. It was found that students in group GA01, who had no previous experience with such controls, were more prone to commit plagiarism across a variety of exercises.

Conversely, the group that had previous experience with plagiarism control measures (group HN01) exhibited markedly lower instances of attempted plagiarism. This distinct contrast between the two groups underscores the importance of regular and consistent enforcement of anti-plagiarism policies. It appears that experience with such measures fosters a culture



of academic integrity among students, leading to decreased incidences of plagiarism.

However, it is important to note that this experiment does not facilitate a direct comparison of results with the final exam. The GA01 group completed the final exam without supervision, whereas the HN01 group underwent a supervised examination. Consequently, it would be unfair to make a comparison between these groups in this particular context.

**3. Initial warning.** The third experiment conducted as part of this study offers compelling evidence about the effectiveness of the early implementation of plagiarism control measures. Mirroring the design of the first experiment but targeting a new cohort of students (group HN02), this investigation revealed an overwhelmingly positive impact of early plagiarism control introduction. Indeed, the instances of similarity in submitted assignments, which could potentially indicate plagiarism, saw a drastic reduction by 96.5 %. This statistically significant finding underscores the preventive power of plagiarism control measures when instituted at the start of the semester, making students acutely aware of the scrutiny their work will undergo.

Furthermore, when comparing student performance across two different academic years (WS21/22 and WS22/23), the data revealed an encouraging upward trend. The overall average grades for exercises witnessed an improvement of 24.6 %.

These findings offer a compelling suggestion that the introduction of plagiarism control mechanisms can act as a powerful catalyst in motivating students to work autonomously. Rather than resorting to shortcuts such as copying others' work, students, when faced with the awareness of plagiarism checks, show a tendency to invest their time and energy in understanding the intricacies of programming, practicing their skills, and enhancing their grasp of the subject matter.

Importantly, the initial notification of plagiarism control did not hinder the formation of study groups, as students were permitted to collaborate during tutor sessions and outside of class (with restrictions on sharing code or solving homework together). Nevertheless, we cannot rule out the possibility that the existence of plagiarism control discourages the formation of study groups. Beyond addressing the primary research queries, this manuscript systematically investigates salient dimensions, encompassing:

**Tool Validation.** The plagiarism control procedure inherently carries the risk of false positive outcomes (as we got in the first experiment in this study), which incorrectly flag non-plagiarized work as a suspect. In response to this issue, systematic bilateral communication is instituted between students and instructors. This dialogue enables students to construct a well-substantiated response that unequivocally demonstrates the error made by the plagiarism control. Additionally, this discussion provides instructors with insights into the students' comprehensive understanding of the subject matter to which the assignment pertains.

**Generalization.** The scope of this article is expressly confined to experimental investigations conducted within courses that comprise the study and implementation of fundamentals

of programming and software engineering. As a consequence, it is not possible to guarantee the generalizability of the results obtained from these experiments to other learning topics.

**Stress Presence.** In WS21/22, one student, who failed the course due to plagiarism, expressed, "I independently completed all the assignments this semester (WS22/23) as I did not wish to fail again. It was not worth repeating the course because I had shared code for two exercises in the previous semester." This indicates that students understand the severe consequences of committing plagiarism. Therefore, they tend to avoid any misconduct practice during the development of independent assignments. Additionally, new students also are aware of this, and they also tend to avoid this misconduct.

## VI. CONCLUSION AND FUTURE WORK

The primary contribution of this article is its provision of empirically backed findings that illustrate the propensity of students to commit to plagiarism when given an opportunity. The data also reveals an elevated likelihood of plagiarism among students that have not previously encountered plagiarism control initiatives in previous courses.

Additionally, this study suggests that the mere act of informing students about plagiarism control measures significantly deters plagiaristic activities. This not only leads to a decrease in plagiaristic instances but also fosters a positive impact on students' overall academic performance as evidenced by their improved results in both assignments and examinations.

Given the deductions drawn from this investigation, it is highly recommended that anti-plagiarism mechanisms be instituted and explicitly communicated to the students for every assignment throughout the course's duration. Evidence from the third experiment further corroborates this recommendation by showcasing how such measures foster independent work ethics among the students.

Moving forward, the validation of these findings with substantial statistical evidence is a critical step. Two-sided significance tests should be conducted once a larger dataset is obtained over a more extended period. Furthermore, it would be of high value to evaluate and compare plagiarism behavior across different academic fields such as computer science and economics. This comparative analysis would enable the identification of potential contributing factors to observed variations in plagiarism behavior. These may include distinct learning and collaboration strategies or unique academic traditions associated with each field of study.

However, plagiarism checking should not be used in every case. When it comes to creativity in project-based courses, a multitude of rules, guidelines, and instructions can limit students' thought processes and thus reduce student learning [26]. It is important to weigh the use of plagiarism control and use it where it makes sense.

In the future, we will implement a continuous plagiarism checker in Artemis that can be activated at the exercise level. It will inform students about possible plagiarism immediately after submission - i.e. while the submission is still in progress.

This way, students will be made aware of the possible consequences as early as possible and will have the opportunity to correct the accusation in a later submission. Since JPlag works on the abstract syntax tree, it will not be sufficient to apply simple refactoring such as automatic renaming of variables. Your solution will have to be inherently different in subsequent attempts to remove the plagiarism allegation. Interviews will reveal the psychological impact on student learning outcomes and how early notification affects their cheating behavior.

## REFERENCES

- [1] I. Albluwi, "Plagiarism in programming assessments: a systematic review," *Transactions on Computing Education*, vol. 20, no. 1, pp. 1–28, 2019.
- [2] M. Joy and M. Luck, "Plagiarism in programming assignments," *Transactions on education*, vol. 42, no. 2, pp. 129–133, 1999.
- [3] E. Luquini and N. Omar, "Programming plagiarism as a social phenomenon," in *Global Engineering Education Conference*, pp. 895–902, IEEE, 2011.
- [4] R. J. Youmans, "Does the adoption of plagiarism-detection software in higher education reduce plagiarism?," *Studies in Higher Education*, vol. 36, no. 7, pp. 749–761, 2011.
- [5] G. Hill, J. Mason, and A. Dunn, "Contract cheating: an increasing challenge for global academic community arising from covid-19," *Research and practice in technology enhanced learning*, vol. 16, pp. 1–20, 2021.
- [6] T. Foltýnek, D. Dlabolová, A. Anohina-Naumeca, S. Razi, J. Kravjar, L. Kamzola, J. Guerrero-Dib, Ö. Çelik, and D. Weber-Wulff, "Testing of support tools for plagiarism detection," *International Journal of Educational Technology in Higher Education*, vol. 17, pp. 1–31, 2020.
- [7] B. D. Jenkins, J. M. Golding, A. M. Le Grand, M. M. Levi, and A. M. Pals, "When opportunity knocks: College students' cheating amid the covid-19 pandemic," *Teaching of Psychology*, 2022.
- [8] N. Tahaei and D. C. Noelle, "Automated plagiarism detection for computer programming exercises based on patterns of resubmission," in *Conference on International Computing Education Research*, pp. 178–186, 2018.
- [9] J. Pierce and C. Zilles, "Investigating student plagiarism patterns and correlations to grades," in *Technical Symposium on Computer Science Education*, pp. 471–476, 2017.
- [10] J. C. Paiva, J. P. Leal, and Á. Figueira, "Automated assessment in computer science education: A state-of-the-art review," *Transactions on Computing Education*, vol. 22, no. 3, pp. 1–40, 2022.
- [11] R. Stallman, "Gnu compiler collection internals," *Free Software Foundation*, 2002.
- [12] J. Hage, P. Rademaker, and N. Van Vugt, "A comparison of plagiarism detection tools," *Utrecht University. Utrecht, The Netherlands*, vol. 28, no. 1, 2010.
- [13] M. Kaya and S. A. Özel, "Integrating an online compiler and a plagiarism detection tool into the moodle distance education system for easy assessment of programming assignments," *Computer Applications in Engineering Education*, vol. 23, no. 3, pp. 363–373, 2015.
- [14] D. Pawelczak, "Benefits and drawbacks of source code plagiarism detection in engineering education," in *Global Engineering Education Conference*, pp. 1048–1056, IEEE, 2018.
- [15] D. Pawelczak, "Effects of plagiarism in introductory programming courses on the learning outcomes," in *5th International Conference on Higher Education Advances*, pp. 623–631, 2019.
- [16] B. Halak and M. El-Hajjar, "Plagiarism detection and prevention techniques in engineering education," in *11th European Workshop on Microelectronics Education*, pp. 1–3, IEEE, 2016.
- [17] S. Moss, B. White, and J. Lee, "A systematic review into the psychological causes and correlates of plagiarism," *Ethics & Behavior*, vol. 28, no. 4, pp. 261–283, 2018.
- [18] S. Krusche, A. Seitz, J. Börstler, and B. Bruegge, "Interactive learning: Increasing student participation through shorter exercise cycles," in *Proceedings of the 19th Australasian Computing Education Conference*, pp. 17–26, 2017.
- [19] S. Krusche and A. Seitz, "Increasing the interactivity in software engineering moocs - A case study," in *52nd Hawaii International Conference on System Sciences, HICSS*, pp. 1–10, ScholarSpace, 2019.
- [20] S. Krusche, N. von Frankenberg, and S. Afifi, "Experiences of a software engineering course based on interactive learning," in *15. Workshops Software Engineering im Unterricht der Hochschulen*, pp. 32–40, 2017.
- [21] S. Krusche, N. von Frankenberg, L. M. Reimer, and B. Bruegge, "An interactive learning method to engage students in modeling," in *42nd International Conference on Software Engineering: Software Engineering Education and Training*, pp. 12–22, ACM/IEEE, 2020.
- [22] S. Krusche and A. Seitz, "Artemis: An automatic assessment management system for interactive learning," in *Proceedings of the 49th technical symposium on computer science education*, pp. 284–289, ACM, 2018.
- [23] S. Krusche, "Interactive learning - A Scalable and Adaptive Learning Approach for Large Courses," Habilitation, Technical University of Munich, 2021.
- [24] L. Prechelt, G. Malpohl, M. Philippsen, *et al.*, "Finding plagiarisms among a set of programs with jplag.," vol. 8, no. 11, p. 1016, 2002.
- [25] S. Engels, V. Lakshmanan, and M. Craig, "Plagiarism detection using feature-based neural networks," in *38th technical symposium on Computer science education*, pp. 34–38, 2007.
- [26] S. Krusche, B. Bruegge, I. Camilleri, K. Krinkin, A. Seitz, and C. Wöbker, "Chaordic Learning: A Case Study," in *39th International Conference on Software Engineering: Software Engineering Education and Training*, pp. 87–96, IEEE, 2017.