Software Framework for Mitigating Programming Plagiarism and Collusion

Oscar Karnalim
Maranatha Christian University, Indonesia and University of Newcastle, Australia
E-mail: oscar.karnalim@it.maranatha.edu, oscar.karnalim@newcastle.edu.au

Thesis summary

Keywords: Academic integrity, learning technology, computing education, code similarity

Received: January 22, 2026

Many mitigation strategies for programming plagiarism and collusion focus solely on either penalising students involved in such misbehaviour or manually educating students regarding the matter. This paper combines both strategies within a software framework and automates the education strategy. It informs students about plagiarism and collusion based on code similarity and instructors' expectations through an assessment submission system with three variants. Highly similar submissions are alerted, while original submissions have their similarities simulated. In addition, the quality of the submissions is also reported through static analysis. On the due date, student submissions are checked using a similarity detector that provides human-language explanations, which has two variants. Students using our framework show greater awareness of programming plagiarism and collusion, and are less likely to engage in such misbehaviours.

Povzetek: Disertacija izobražuje in razvija podporo proti plagiatorstvu.

1 Introduction and method

In programming, plagiarism and collusion refer to reusing one's work without proper acknowledgement, and the only difference is that, in collusion, the original owner(s) allow it [3]. There are several strategies to mitigate such misbehaviours, yet most of them focus on penalties (e.g., developing similarity detectors for plagiarism and collusion) [2]. Few of them focus on educating students on the matter, but most are conducted manually by instructors [4], which can be labour-intensive.

This paper presents a software framework to mitigate programming plagiarism and collusion by combining both educational and punitive strategies and automating the educational one. It consists of an assessment submission system (where students submit their work) and a similarity detector for plagiarism and collusion. Currently, the framework caters for Java and Python.

The assessment submission system generates an online report and sends it to the student upon each submission. The report contains instructors' expectations regarding plagiarism and collusion in their courses, as well as certain information on code similarities. To promote the relevancy, code similarities are simulated from the submission by applying superficial disguises to uncommon code segments. For submissions that are highly similar to others, students will be alerted and warned without revealing their identities to one another. Instructors will also be informed to apply appropriate measures.

The assessment submission system has three variants. The short segments variant reports short similarities (even two program statements), putting minimal pressure on students involved in plagiarism and collusion, since some of these similarities may be coincidental. The long segments variant reduces coincidental similarities by focusing on longer similar code segments (at least 8 program statements). It puts more pressure on students involved in plagiarism and collusion, as their submissions will be suspected, while others' are not. The gamified long segments variant expands the long segments variants through gamification. Students can earn more game points by submitting unique programs early and earning badges. Students' progress will be displayed in a leaderboard, and top students are commonly incentivised. To further promote student engagement, the quality of their code is reported through static analysis.

The similarity detector for plagiarism and collusion is applied to all student submissions for each assessment. The submissions will be compared pairwise, and highly similar submissions will be manually checked for plagiarism and collusion. Students whose submissions are likely to result from such misbehaviour will be penalised in accordance with the policies.

The similarity detector has two variants. The standard variant is dedicated to small assessments where the expected solutions share the same semantics. Such assessments are quite common in programming. The variant focuses on syntactic rather than semantic similarity. Each submission is converted to a token string with comments and white space removed. All identifiers, constants, and data types are generalised to their own types. The token strings are then compared using the running Karp-Rabin

greedy string tiling algorithm, a common approach for automated detectors. Any reported similarities will be featured with human language explanation. The comprehensive variant can cater for a broad range of assessments at the expense of processing time. It generates three reports at once, covering semantic, syntactic, and surface similarities. Instructors can integrate findings obtained from those reports to identify plagiarism and collusion.

2 Evaluation

The software framework was evaluated through three quasi-experiments. The first experiment (163 students) measured the impact of the short segments variant by comparing it with the baseline approach (without using the software framework). The second experiment (202 students) measured the impact of the long segments variant compared with the short segments counterpart. The third experiment (240 students) measured the impact of gamification on the long segments variant. According to the experiments, students in the short segments variant were more aware of plagiarism and collusion and were less likely to engage in such misconduct. The long segments variant appeared more effective, and the gamified variant promoted student engagement.

Both similarity detectors were evaluated based on hundreds of student submissions. They were more effective than JPlag, a common similarity detector. Several additional experiments were conducted to support the study. They covered many aspects, including preprocessing steps, token representations, common code removal, code quality recommendation, gamification incentives, and other technical modules.

3 Conclusion

This paper presents a software framework to mitigate programming plagiarism and collusion by integrating an assessment submission system with a similarity detector. Students with the software framework are more aware of the matter, resulting in less engagement in such misbehaviour.

Acknowledgement

The author would like to thank all students and instructors involved in the studies at Maranatha Christian University, Indonesia. The paper is a summary of the author's PhD thesis [1], which was awarded the Rob Reilly Best Doctoral Dissertation in Engineering Education Award by the IEEE Education Society in 2024.

References

[1] Karnalim, O. *Building awareness of programming plagiarism and collusion through similarity feedback generation*. PhD thesis, University of Newcastle, Australia, 2022.

[2] Novak, M., Joy, M., and Kermek, D. Source-code similarity detection and detection tools used in academia: a systematic review. *ACM TOCE 19*, 3 (2019), 27:1–27:37.

[3] Parthasarathy, P. D., Kapoor, I., Joshi, S., and Thomas, S. Influence of personality traits on plagiarism through collusion in programming assignments. In *ACM ICER* (2024), ACM, p. 143–153.

[4] Simon, Sheard, J., Morgan, M., Petersen, A., Settle, A., and Sinclair, J. Informing students about academic integrity in programming. In *ACM ACE* (2018), ACM, pp. 113–122.