Plagiarism in Programming Assignments

Hariprasath Govindarajan (hargo729)

Abstract

Plagiarism is a troubling occurrence which has been rising in educational institutions. Plagiarism in programming assignments are much different from text-based assignments and they are not easy to detect. Since programming assignments are a key component of several curriculums and a place to gain valuable practical experience, it is a significant part of a student's learning process. So, it is of paramount importance to prevent plagiarism in the context of programming. This paper explores student perspectives regarding plagiarism in programming to better understand the reasons leading to it and examines different approaches to curb it.

Introduction

Plagiarism has gradually grown to become a common term in universities among students and faculties. It has become a major issue plaguing education as indicated by reports of increasing instances of plagiarism (Hammond 2002). Vast amount of information readily available and accessible on the internet has made it easier and effortless for students to plagiarize. This is not just limited to textual assignments and is prevalent in programming assignments as well. Parker and Hamblen (1989) defined a plagiarized program as "a program which has been produced from another program with a small number of routine transformations". In computing courses, a lot of knowledge stands to be gained from practical experience. Programming assignments are a key method to help students to become proficient in programming as well as a vital indicator for professors to assess the students. Source code plagiarism significantly hampers the teaching-learning process and it is very important to gain a better understanding of plagiarized programs in order to design methods to successfully mitigate it. Hence, it is essential to examine the reasons behind students resorting to source code plagiarism.

Typically, educational institutions use text similarity-based plagiarism detection tools like Turnitin for assignment submissions, but I think that they are not effective at detecting plagiarized programs. The lexical elements of a program could be modified while retaining the structure and logic of a copied source code to cheat a text similarity-based plagiarism detection tool. In contrast to essays, this makes source code plagiarism more difficult to detect. While there are tools like JPlag and Moss which abstract the lexical elements and compare the structures of programs, they are limited to comparisons within a finite set of submissions (Sraka and Kaucic 2009). Such tools could be useful to detect programs copied within a class but do not help in deterring students from resorting to online sources. On the other hand, there is a lack of understanding about what constitutes as source code plagiarism among students and a large proportion of suspected plagiarism cases are unintentional (Nadelson 2007). In this paper, we will examine student perspectives towards plagiarism in programming, critically analyze measures suggested in previous studies and propose improvements for the same.

Student attitudes and perspectives

In text-based assignments, there are widely accepted standards to acknowledge the source through citations, paraphrasing with references and adding quoted text as-is. But in programming, clear standards regarding attribution are lacking. On the other hand, reuse of existing functionality is often encouraged in programming. This creates a confusion among students regarding what is acceptable and not acceptable behaviour and leads to unintentional plagiarism. Based on a focus group based research, Simon et al. (2013) observed how students decided to do things based on their individual ethics when the boundaries of acceptable behaviour were unclear. Students tend to be doubtful about the extent of reuse that is acceptable - if only ideas can be reused and how much source code can be reused. I think that instilling a clarity of standards to be observed in a course or programme and defining a code of ethics as per the course objectives and customized to reflect the realities of programming would help in minimizing unintentional plagiarism.

Students sometimes struggle with programming assignments, lose confidence in their ability to meet the submission deadline and resort to just copying from other students or sources from the internet. From my point of view, a healthy and positive way to curb this is to encourage collaboration and exchange of ideas with other students. Like it is done at Linkoping university, where I study, this can also be planned as a formal practice in the course schedule through seminars held during the submission period of an assignment. During this process, it is essential for students to learn to take another person's ideas as purely an inspiration to build their own novel approach. However, discussing an assignment with fellow students can sometimes make the resultant submissions more similar within the group of students. Bradley (2016) found that the divergence/convergence of assignment solutions had a significant effect on the natural similarity in students' submissions. In the scope of programming assignments, convergent assignments have only one solution approach while divergent assignments have multiple solution approaches. Divergent assignments would result in more varied solutions and ensure that even when students discuss ideas, they would be able to come up with different approaches on their own. Additionally, I think that conducting a review of the variety of solution approaches would help the students to better understand certain nuances and improve their problem-solving process.

When students plagiarize intentionally, they fail to develop their skills to expected levels. In my view, this makes it difficult for them to strive in the workplace and similar behaviour at the workplace could lead to more serious consequences in industries and businesses. When alumni of certain educational institutions are found to be less skilled or lacking integrity, it undermines the reputation of the institutions as well. Hence, it is highly important to detect intentional plagiarism and take corrective measures to curb its occurrence among students.

Automated source code plagiarism detection

Intentionally plagiarized programs are often highly obfuscated from the source program and text-based plagiarism detection tools are not suitable for detecting plagiarism in such content. Some automated plagiarism detection software do exist for detecting plagiarized programs as well. Modiba, Pieterse, and Haskins (2016) analyzed several plagiarism detection software for programming assignments like AC, CodeMatch, CPD, MOSS, NED and ANALYSIS. Each of them was applicable to a specific set of programming languages. Only CodeMatch, MOSS and to a certain extent, CPD were capable of handling a wide variety of programming languages. NED, which was the best performer in their study is not available for public use and hence, will not be discussed further. All the others had an accuracy of less than 70% at source code plagiarism detection. From my point of view, minimizing false positives is an important factor to ensure that the process is ethically sound. In that aspect, MOSS appeared to have performed the best. Different tools are efficient at detecting different obfuscation methods and the authors concluded that the most suitable one for a course should be decided on a case-by-case basis. Also, these tools could only be used as an extension and not as a completely unsupervised substitute to the entire plagiarism detection process.

Due to the reliability issues with plagiarism detection tools for programming, manual checking appears to be necessary for effectively assessing submissions. This is a time-consuming process with logistical concerns but can be streamlined to design a balanced approach satisfying the temporal constraints. One such approach was proposed by Heron and Belford (2016) which also accounts for ethical considerations. One of the plagiarism detection software needed to be chosen based on the programming language and by testing the software for the specific course. This automated plagiarism detection would be used as a preliminary step to identify possible suspects. There might exist some well obfuscated plagiarized submissions which would get through if we only checked the suspicious ones. Hence, it was proposed that a certain number of submissions which present no reason to suspect be included along with the suspicious submissions for thorough checking. Even if these are genuine submissions, they would serve as a benchmark while assessing plagiarism in the suspect submissions.

I think that the above process would ensure a fair and reliable evaluation and would allow the evaluator to decide the number of assessments to be checked manually based on availability of time. In the survey carried out in Simon et al. (2013), the reasons because of which students believed that they can get away with plagiarized programs was either that they thought that there was only one solution or that they expected the teacher to just run the programs and not examine their originality or a combination of both. I strongly believe that having an explicit process in place like the one explained above would serve as a deterrent against plagiarism to students. This is partially evidenced in the study by Pawelczak (2018), where improvement in performances in examinations were observed after the introduction of a plagiarism detection system.

Plagiarism resistant grading system

Plagiarism mitigation policies involving plagiarism detection checks, disciplinary actions and punishments operate from a negative connotation by creating fear in the minds of students. Often, students forget about developing skillsets in the long run and focus on the here-and-now aspects of passing the course by any means necessary. A more positive approach could be undertaken by means of the grading system. Placing more emphasis on the importance of genuinely learning concepts in achieving higher grades in the course would encourage students to take programming assignments much more seriously. In this way, their short-term goals would also be directed towards mastering the course content.

Grades are expected to reflect a student's proficiency in a course and many programming-based courses provide a sizeable amount of weightage to programming assignments while calculating the grades. To maintain a fair grading system, it is all the more vital to ensure academic honesty in the programming assignments. Hwang and Gibson (1982) state that a positive prevention strategy against plagiarism using grading methods would be an effective way to curb plagiarism in the long run. They analyzed some of the commonly used grading methods by comparing their benefits and drawbacks. Varying the weightage given to programming assignments and final examination appeared to have different disadvantages. On the other hand, a blended approach where programming assignments were followed by a closely related quiz appeared to be more beneficial.

For a blended approach, Hwang and Gibson (1982) suggested two aggregation methods for scores in the assignment and the quiz. The quizzes should be framed based on the programming assignment alone and in order to do well in the quiz, the student needed to understand and carry out the programming assignment by themselves. The score for the programming assignment could be arrived at as:

1. The product of the percentage scores in the quiz and the assignment
2. The sum of the scores in the quiz and the assignment.

On further analysis it was apparent that the grades calculated using the sum of the scores was well balanced between deterring dishonest students and rewarding the honest ones. However, sometimes an honest and well performing student could have had a bad day in a quiz and got a poor score due to unexpected circumstances. I suggest that such occurrences could be minimized by considering only the best *(n-1)* assignment scores out of *(n)* number of assignments for grading each student. This way, the effects of such unforeseen events on student performance in a quiz can be minimized.

Conclusion

Programming assignments are a vital part of a course curriculum and shape the learning curves of students. If left unchecked, plagiarism in programming would greatly affect the amount of knowledge gained by students during their academic studies. Understanding plagiarism and the reasons for its occurrence in the context of programming is very essential to build methods to curb it. In this paper, we explored the student perspectives regarding plagiarism and explained some of the factors that lead to it. In order to dispel any confusion among students and avoid unintentional plagiarism, it is key to have well defined academic standards and policies for courses. Better collaboration among students facilitated by interesting assignments that have divergent solutions would help students to overcome their learning struggles.

If plagiarism is done intentionally, it is critical to detect it early and take corrective measures to instil academic integrity. Though there are various software to detect plagiarism in programming, each have their own merits and demerits and none of them have reached the level of maturity to be used in a fully unsupervised manner and as a substitute to human processes. A fair and efficient plagiarism detection system is necessary to serve as a deterrent for plagiarism. However, a suitable software could be integrated into the plagiarism detection process to preliminarily identify suspect submissions easily. Human verification involving analysis of more submissions would provide a chance to detect well obfuscated plagiarized submissions also.

An approach based on the grading system to preventing plagiarism could have a more positive influence on students. Blending programming assignments with immediate quizzes based on the assignment would be a fairer way to test the concepts and knowledge learned by students. Better grades are a great motivator that would encourage students to work harder and to complete programming assignments on their own. Hence, through a combination of clear standards, well defined plagiarism detection system and a grading system that is resistant to plagiarism, I believe that we could comprehensively reduce the occurrence of plagiarism to a large extent. This is something that every higher education institution should strive to achieve in order to foster a more prosperous learning environment for students. On the other hand, students should realize how plagiarism harms their own academic progress and uphold the academic integrity which would enable them to launch successful careers.

References

Bradley, Steven. 2016. "Managing Plagiarism in Programming Assignments with Blended Assessment and Randomisation." In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, 21–30. Koli Calling '16. New York, NY, USA: ACM. doi:10.1145/2999541.2999560.

Hammond, Michael. 2002. "Cyber-Plagiarism: Are Fe Students Getting Away with Words?" Education-line.

Heron, Michael James, and Pauline Belford. 2016. "Musings on Misconduct: A Practitioner Reflection on the Ethical Investigation of Plagiarism Within Programming Modules." *SIGCAS Comput. Soc.* 45 (3). New York, NY, USA: ACM: 438–44. doi:10.1145/2874239.2874304.

Hwang, C Jinshong, and Darryl E Gibson. 1982. "Using an Effective Grading Method for Preventing Plagiarism of Programming Assignments." *ACM SIGCSE Bulletin* 14 (1). ACM: 50–59.

Modiba, Phatludi, Vreda Pieterse, and Bertram Haskins. 2016. "Evaluating Plagiarism Detection Software for Introductory Programming Assignments." In *Proceedings of the Computer Science Education Research Conference 2016*, 37–46. CSERC '16. New York, NY, USA: ACM. doi:10.1145/2998551.2998558.

Nadelson, Sandra. 2007. "Academic Misconduct by University Students: Faculty Perceptions and Responses." *Plagiary: Cross Disciplinary Studies in Plagiarism, Fabrication, and Falsification* 2 (2).

Parker, A., and J. O. Hamblen. 1989. "Computer Algorithms for Plagiarism Detection." *IEEE Transactions on Education* 32 (2): 94–99. doi:10.1109/13.28038.

Pawelczak, D. 2018. "Benefits and Drawbacks of Source Code Plagiarism Detection in Engineering Education." In *2018 Ieee Global Engineering Education Conference (Educon)*, 1048–56. doi:10.1109/EDUCON.2018.8363346.

Simon, Beth Cook, Judy Sheard, Angela Carbone, and Chris Johnson. 2013. "Academic Integrity: Differences Between Computing Assessments and Essays." In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, 23–32. Koli Calling '13. New York, NY, USA: ACM. doi:10.1145/2526968.2526971.

Sraka, D., and B. Kaucic. 2009. "Source Code Plagiarism." In *Proceedings of the Iti 2009 31st International Conference on Information Technology Interfaces*, 461–66. doi:10.1109/ITI.2009.5196127.