Plagiarism Deterrence in CS1 Through Keystroke Data

Kaden Hart
Utah State University
Logan, Utah
kaden.hart@usu.edu

Chad Mano
Utah State University
Logan, Utah
chad.mano@usu.edu

John Edwards
Utah State University
Logan, Utah
john.edwards@usu.edu

Abstract

Recent work in computing education has explored the idea of analyzing and grading using the process of writing a computer program rather than just the final submitted code. We build on this idea by investigating the effect on plagiarism when the process of coding, in the form of keystroke logs, is submitted for grading in addition to the final code. We report results from two terms of a university CS1 course in which students submitted keystroke logs. We find that when students are required to submit a log of keystrokes together with their written code they are less likely to plagiarize. In this paper we explore issues of implementation, adoption, deterrence, anxiety, and privacy. Our keystroke logging software is available in the form of an IDE plugin in a public plugin repository.

CCS Concepts

• **Social and professional topics** → CS1.

Keywords

CS1, Keystrokes, Plagiarism

ACM Reference Format:
Kaden Hart, Chad Mano, and John Edwards. 2023. Plagiarism Deterrence in CS1 Through Keystroke Data. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023), March 15–18, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3545945.3569805

1 INTRODUCTION

Universities have reported that computer science departments are the worst offenders of academic integrity code violations [18]. Plagiarism detection tools such as MOSS have been deployed to catch instances of cheating, but many students think they can circumvent these detection tools and are not deterred. Catching plagiarism after it has occurred is necessary for universities to do, but is too late to help students. Students who are caught plagiarizing at our university receive an immediate failing grade for the course and face expulsion in serious or repeated cases.

Recent scholarship has discussed the benefits of collecting programming process data such as keystrokes, compiles, saves, and commits [e.g. 10]. The majority of collecting such data has been for research purposes, but calls for collecting and using it for identification of at-risk students, assessment, and awareness are increasing. A highly suitable application of this type of data is plagiarism detection, though it has been little talked about. However, the usefulness of process data, especially keystrokes, in detecting and deterring plagiarism can hardly be questioned. It offers a fine-grained view of the process a student took to complete a computer programming assignment.

In this experience report we discuss a two-semester project where students in our CS1 classes installed the ShowYourWork plugin that logged their keystrokes to a file on their computer. One group of students were required to submit the keystroke file along with their submitted code. We discuss the effect of simply installing the keystroke logger in deterring students from plagiarism. We report alleged cases of plagiarism (detected using standard code checkers and not the keystroke logs), answers to a series of Likert-style questions, and responses in a free-response paragraph in which students describe their experience with keystroke logging. We find that simply logging keystrokes appears to deter plagiarism through a reduced temptation to cheat and that most students feel positively about the tool. Unintended side-effects are an increased anxiety level of some students due to fears that they will be falsely accused of cheating and a sense that the instructor didn't trust them. We also discuss issues of privacy.

SIGCSE 2023, March 15–18, 2023, Toronto, ON, Canada

ACM ISBN 978-1-4503-9431-4/23/03.
https://doi.org/10.1145/3545945.3569805

2 BACKGROUND

2.1 Plagiarism Detection and Shortcomings

There are many ways to detect plagiarism, and students find methods of circumventing each of them. Code similarity detection software such as MOSS can detect if a student copied code from somewhere else by inspecting the final submission of code [20]. MOSS can be defeated by students who take sufficient care in renaming variables and shuffling lines of code around. Comparing the abstract syntax trees [7], or binary compilations [16], of programs can still detect plagiarism even with renaming of variables and small changes to the order of lines of code; however, a student who makes larger changes to plagiarized code, especially to the order of conditionals, may evade detection.

Other methods to detect plagiarism have been used such as signing documents with white space characters that will be copied if students share their work [8], or the use of version control systems [17] to track the history of a document. Others have used keystroke data to authenticate the author of a document by their typing behaviors [13, 15]. While keystroke data provides rich information about the creation of a document, it is rarely available because it requires students to use online editors that track interactions, or have software installed on their development environments that will log their keystrokes.

Ahadi et. al. [2] compared three popular plagiarism detection tools – MOSS, SIM, and JPlag – and found that they do not always

agree on what is potential plagiarism and what isn't. Ahadi et. al. also found that the software packages give many false positives for plagiarism. False positives can be especially common in CS1, where assignments can be so small that code similarity checkers can flag almost every submission as possible plagiarism. Instructors in our university have reported that they do not check for plagiarism on some assignments because the assignments were too small to determine if plagiarism occurred or not.

Not only do plagiarism detection tools not agree on what is plagiarism and what isn't, students and instructors don't agree either. Stepp et. al. [21] found a lack of consensus among students on what constituted cheating, and suggested that discussion of what is and is not cheating could be useful for students. Brimble et. al. [5] gave 1206 students and 190 academic staff 20 scenarios about academic integrity, and for every scenario, students rated seriousness and penalty lower and prevalence higher than the staff.

2.2 Programming Process

Recent scholarship has investigated analysis of the process of writing a computer program using log data. For example, Jadud et al. [11] proposed a metric called the error quotient (EQ), based on compile behavior, to measure probability of success of a student. Similar metrics include the Watwin score [23], the RED score [4], and NPSM [6]. Similar metrics from process data include number of attempts of programming exercises [1], code submissions [22], and typing speed [14], among others. Ihantola et al. [10] give an excellent overview of programming process data, characteristics of the different types, and examples of uses.

Despite the demonstrated utility of collecting programming process data, it does not seem to have been adopted in an educational (as opposed to research) setting. To our knowledge, no CS1 course captures keystrokes as an assessment or even as an observational tool. We are aware of classrooms that collect more course-grained data, however. In classes at our university, assignments can be submitted via a repository like GitHub or Bit Bucket. These classes require students to have at least a certain number of commits, quality commits, and good commit messages, and the student's grade is determined by how well they followed good coding practices in addition to submitting a quality final product. Similarly, in software engineering classes, students are typically required to submit process documents such as burndown charts, reports of standup meetings, and user stories. Despite a rich history of course-grained process data collection, educators have not seemed to have adopted richer, fine-grained data.

3 METHODS

This paper reports on two consecutive semesters (14-week terms) of a CS1 course taught at a U.S. university: Fall 2021 and Spring 2022. Three groups of students, one whose keystrokes were not logged, one whose keystrokes were logged but not made available to the instructor, and one whose keystrokes were made available to the instructor. See Table 1. In both semesters students were warned about the consequences of plagiarism and notified that the instructor did their best to identify and prosecute cases of plagiarism. In both semesters (and semesters prior as well) the instructor used MOSS to detect possible instances of plagiarism.

Group	Semester	Keystrokes logged	Known to instructor	*n*
no logging	Fall 2021			200
unsubmitted logging	Fall 2021	x		62
submitted logging	Spring 2022	x	x	245

Table 1: Details of three groups of students. *n* is the number of participants.

All students were given a survey at the end of the semester. Both semesters of our study were conducted according to a protocol approved by our university's ethics review board.

3.1 Keystroke Data Collection

In the first semester, students were given the opportunity to opt into a study which included installing the ShowYourWork plugin to their IDE that logged their keystrokes. No compensation was offered for opting in. Keystrokes were available only to researchers and not the instructor and students were informed of this fact. We call these students the *unsubmitted logging* group. See Table 1. Students in the first semester who chose not to participate in the study did not have their keystrokes logged and are called the *no logging* group in this paper. In the second semester all students were required to install the keystroke logger plugin and all keystrokes were submitted to the instructor. These students compose the *submitted logging* group.

ShowYourWork is a PyCharm IDE plugin that logs keystrokes to a local file in a sibling directory (*unsubmitted logging* group) or the same directory (*submitted logging* group) as the source code. Students in the *submitted logging* group were required to submit the keystroke logs together with their code submissions. The plugin collected two types of data: keystrokes, and edits to project files. With keystroke and file edit data, the step-by-step process students made to write their code can be recreated.

3.2 Privacy

An important issue with keystroke logging is privacy. Knowing that privacy was a concern we discussed the issue with our university's general counsel during development of ShowYourWork. The attorney agreed that capturing keystrokes was analogous to and even literally having students show their work. The attorney found that our approach is ethical given the following conditions:

- Students are informed that their keystrokes are being recorded, including which files the logs are being written to.
- Students are informed that they have control over their keystroke logs and that no university faculty or staff will see them until the student submits them with their assignment.
- Students are informed that only keystrokes within PyCharm are recorded.
- Recorded keystrokes while working on computer programming projects not related to CS1 remain privately stored on their computer.
- Students are informed what the keystrokes will be used for, namely, discussions with instructors/TAs, plagiarism detection, improvement of instruction, and estimation of effort.

- Keystrokes are not used for any other purpose other than those stated above.

Citing privacy concerns, some faculty members were opposed to requiring keystroke logging for CS1. But with the above conditions, our department voted to proceed with requiring the plugin.

An important part of these conditions is student access to their keystroke data. Students own and can inspect their keystroke data, either directly in the log file or through the playback feature, to know what is or is not logged. Students can even modify and remove parts of their logs. However, doing so is extremely difficult without corrupting the file such that reconstruction and playback result in jumbled code and out of bounds errors. However, during the course of the semester we realized that we need to provide students with a tool to delete events from the log. This is necessary because students may inadvertently type something sensitive, such as an accidental paste of a credit card number or password. Furthermore, providing students with a tool to expunge keystrokes could inspire confidence that the keystroke logger needn't be feared. Of course, such a tool could be used to mask plagiarism. However, that risk is attenuated because most students would realize doing so will appear suspicious.

Courses that grade the coding process as part of the assignment do not receive complaints about privacy. Students have been required to use Git, and have a certain number of commits. Git commits show file states similarly to keystroke data, keystroke data is simply higher-resolution, and automatically done with every keystroke.

3.3 Step-By-Step Playback

In Spring 2022 ShowYourWork was updated to include a step-by-step playback of assignments. After installing the plugin, an additional window is available in PyCharm that has a text section and a slider. The text section shows the state of the focused file in PyCharm at the selected position of the slider. The slider has a position for every edit done to the current focused file in PyCharm. Students can start from any point in the edit history of a file and see step-by-step edits to any other point in the file's history. This playback feature used the keystroke log to build code snapshots to show during playback. It was added as a convenience to the students. The playback feature is not a part of the plagiarism study reported in this paper, but we mention it here because two questions in the survey ask about its usefulness to students.

3.4 Survey

At the end of both semesters students were given the opportunity to respond to our survey about plagiarism. Responses can be used to determine how keystroke logging affects student attitudes, potentially reducing plagiarism. Surveys included the following prompts, which used a 5-point Likert scale with response options *Strongly disagree*, *Somewhat disagree*, *Neither agree nor disagree*, *Somewhat agree*, and *Strongly agree*:

- I was tempted to plagiarize in CS1 this semester.
- I believe that I would get caught if I plagiarized in CS1.
- If I copied someone else's code, I would be capable of making it look like I didn't copy.
- The CS1 instructor is capable of finding cases of plagiarism.

Code	Kappa	%	Description
privacy	0.65	6.4%	Invaded privacy
forgot	0.86	11.1%	"I forgot about it"
remembered	0.71	6.4%	"It was on my mind"
no trust	1.0	1.6%	Instructor doesn't trust students
anxiety	0.40	9.5%	Gave anxiety
temptation	1.0	4.8%	Reduced temptation to cheat
positive	0.51	11.1%	A positive thing
negative	0.31	6.4%	A negative thing

Table 2: Coding results. "Kappa" is the inter-rater reliability Kappa values for topics found in free response answers. The complete description for "Gave anxiety" is "Gave anxiety of being accused of cheating even if student wasn't plagiarizing." The % column is the percentage among the 63 responses that mentioned the code. Responses that were empty, "N/A", "none", or similar were ignored.

Students who installed the keystroke logger were additionally asked:

- The plugin that logs keystrokes would make it easier for the CS1 instructor to find cases of plagiarism.
- I would have been more tempted to plagiarize if I hadn't been required to submit the keystroke log file.
- I forgot about the plugin.
- I think the plugin was a good idea.

Students in the second semester (*submitted logging* group) were also asked:

- I used the plugin playback feature.
- The plugin playback feature was useful.
- Please share any additional thoughts you have about the plugin.

Three questions did not use the 5-point Likert scale: "I forgot about the plugin" had the options "Never", "Within 2 months", "Within 2 weeks", and "Immediately", "I used the plugin playback feature" had the options "0 times", "1 to 3 times", "4 to 10 times", and "more than 10 times", "Please share any additional thoughts you have about the plugin" was a free response question.

3.5 Coding

Coding proceeded using the method of Saldana [19]. First, the research team met to create a list of codes, or themes, contained in the survey responses. The result of the discovery session was the list of codes in Table 2. The codebook contained the code name and description.

To code the student responses, two authors used the codebook and independently coded the responses. The codes were compared using Cohen's kappas [3] and pooled kappas [9] to determine inter-rater reliability. The pooled kappa was 0.69, which is considered moderate agreement [12]. For the individual codes' Cohen's kappas, the lowest was 0.31 and the highest was 1.0, or complete agreement (Table 2).

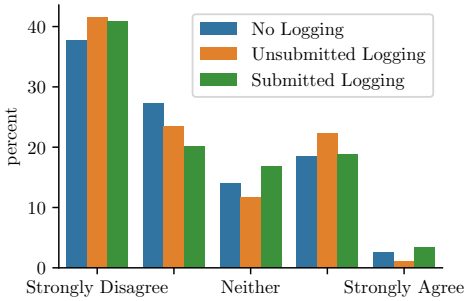

Figure 1: Student responses to "I was tempted to plagiarize in CS1 this semester".

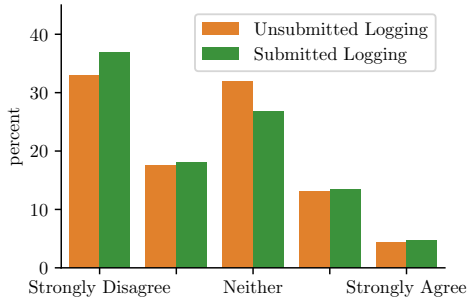

Figure 2: Student responses to "I would have been more tempted to plagiarize if I hadn't been required to submit the keystroke log file".

3.6 Detecting Plagiarism With Keystroke Data

Keystroke data can be used to detect plagiarism in the following cases: (1) copying then renaming variables and shuffling lines of code in PyCharm, (2) copying, renaming, and shuffling outside PyCharm, and (3) retyping plagiarized code. In the first case, if a student copies code, then changes variable names, and moves lines of code around, keystroke data captures the paste event and the original pasted code. The original pasted code can be inspected to determine where the student got the code from. In the second case, if a student copies code and changes it outside the editor then pastes it into their editor, the paste event is stored, and the pasting of a large amount of code is suspicious and can be considered cheating if students have been asked to do all their work in the editor. In the third case, if a student copies code by typing it instead of pasting it, the step-by-step playback will show a student writing code from top to bottom in a single pass; something that rarely happens even with experienced programmers.

The instructor was given a script that would detect large pastes among a set of submissions. It was decided that for this first semester using keystroke logging that the instructor would use keystrokes only for a first pass at plagiarism detection, but would rely on traditional methods for confirmation. As this paper is about deterrence, we defer detailed discussion of plagiarism detection to future work.

4 RESULTS AND DISCUSSION

We received 217 survey responses from the Fall 2021 semester and 155 from the Spring 2022 semester. Of the 155 responses in the spring semester, 63 wrote in the free response question (excluding "none", "N/A", etc.). Information about plagiarism was supplied by a single instructor for their class alone.

4.1 Temptation

When asked, "I was tempted to plagiarize in CS1 this semester", one in five students responded that they strongly or somewhat agreed (Figure 1). Those students are at-risk for plagiarizing, and could benefit from keystroke data requirements. When asked, "I would have been more tempted to plagiarize if I hadn't been required to submit the keystroke log file", 18% of responses reported that requiring keystroke logs reduced their temptation to plagiarize (Figure 2). This number is higher than we expected; nearly 1 in 5 students reporting a reduced temptation to cheat is remarkable.

Semester	Enrollment	Cases	%
Summer 2020	66	13	19.7%
Fall 2020	120	8	6.7%
Spring 2021	223	7	3.1%
Fall 2021	41 | 62	6 | 2	14.6% | 3.2%
Spring 2022	137	5	3.6%
No logging	450	34	7.5%
Logging	199	7	3.5%

Table 3: Cases of plagiarism in CS1 for one instructor from Summer 2020 to Spring 2022. The first value in Fall 2021 is the *no logging* group and the second value is the *unsubmitted logging* group. The % column is the percentage of students that plagiarized.

In the free response section of our survey, 4.8% of students reported that requiring keystroke data reduced their temptation to cheat. One student expressed how requiring keystroke logs reduced their temptation to look at outside resources – something they were asked not to do.

> It definitely did make me not even really look for other code outside of the class to find ways around issues though. i didn't want to get ideas from somewhere else and get accused of cheating even if it was just subconscious from seeing another solution before.

Another student expressed how requiring keystroke data reminded them not to copy code.

> I honestly did not put to much thought into the fact that it was there, but I did always deep down know it was there and figured it could tell I copied code.

Temptation reduction appears to be a benefit of keystroke logging and a deterrent to plagiarize.

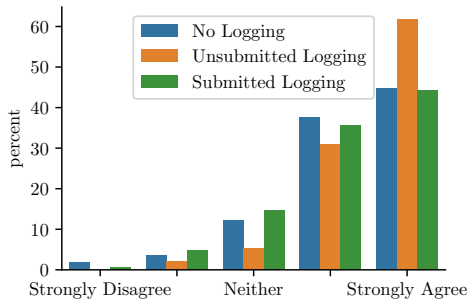

Figure 3: Student responses to "The CS1 instructor is capable of finding cases of plagiarism".

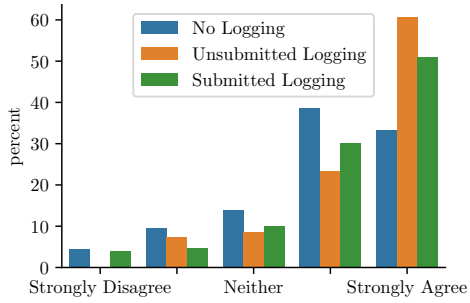

Figure 4: Student responses to "I believe that I would get caught if I plagiarized in CS1".

4.2 Deterrence

As a measure of deterrence, we report prosecuted cases of plagiarism. One of the two instructors participating in our study did not check submissions for plagiarism so we did not use data from his course. We collected plagiarism counts from the other instructor's CS1 courses from Summer 2020 to Spring 2022 in which he used the MOSS code checker to check for plagiarism. See Table 3. While we did see a reduction in plagiarism cases when keystroke logging was in place, it is not a strong result ($z = 1.95, p = .051$). With results from the survey however, we find that requiring keystrokes with assignments increased students' beliefs that they would get caught if they did cheat (Figure 4). We suggest that students who believe they could be caught will be less likely to cheat, and as discussed in Section 4.1, we find a number of students who expressed that they were deterred from cheating.

An interesting dichotomy we discovered in our survey is that students overwhelmingly believe that the instructor is capable of finding cases of plagiarism (Figure 3) and will catch them if they plagiarize (Figure 4), yet students think they are capable of making it look like they didn't plagiarize (Figure 5). Two students' responses to the free response question expressed this "I could if I tried" attitude.

> I think that the plug in was helpful in keeping me honest in my coding. However, if I wanted to plagiarize code I could do it pretty easy.

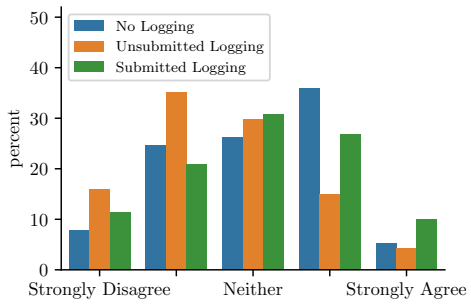

Figure 5: Student responses to "If I copied someone else's code, I would be capable of making it look like I didn't copy".

> It didn't mean much to me. I think that if someone is copypasting plagiarizing, it could be helpful and should discourage that, but if someone is plagiarizing even mildly intelligently, that it wouldn't do much to deter that.

Whether or not we should be concerned that students still think they can cheat is a question we don't know the answer to. On one hand, it seems harmless in the cases of the quoted students, yet it could suggest that students who are prone to cheating are also confident in their ability to mask it.

4.3 Anxiety

False positives are common in plagiarism detection [2]; Keystroke data not only provides better evidence for accusations of plagiarism, but it can also be used to detect false positives. A student falsely accused of cheating could use their keystroke log to defend themselves. Students in our classes may not have understood that keystroke logs can actually work to their benefit. In the free response section of our survey, 9.5% of students expressed anxiety about issues with plagiarism detection through keystroke data. One student in particular had a strong response to the use of keystroke data.

> "Even though I didn't cheat, I was constantly stressed out that my assignments were being marked for plagiarism. This caused A LOT of anxiety throughout the semester, so much so that I wish I didn't take this class because of the emphasis on cheating....It would be really helpful if my professor was more clear about what the plugin is, especially for students who are prone to feelings of anxiety."

This student unfortunately felt significant anxiety, which is understandable. But it is instructive and encouraging that they suggested that becoming more educated about the logging plugin and, in our minds, understanding better the features of a case of plagiarism, would reduce their worry. Other students felt similarly, although not as strongly: "It kind of freaked me out a little bit to be totally honest. But I also forgot about it really quick as well."

Initial concern followed by indifference seems to be a somewhat common theme. It reveals a difference in students, with some dwelling on concerns and others setting them aside rather quickly.

Upon reflection, one of the most important tasks in implementing a measure like this is education and satisfying students that they will not be falsely accused.

> I didn't know much about it except that it recorded every keystroke I made. Sometimes I put together code in a separate test project and pasted it in to my assignment and I was worried that might make it look like I was plagiarizing but it was okay.

Addressing the case of students writing test code outside of the IDE and then pasting it into their own code is an important and difficult use case to consider. One solution is to teach students how to write bits of experimental code within their project. Another is to have them make sure their keystrokes are logged while writing the snippets and keeping the logs around just in case.

Our work seeks to deter plagiarism, but had the unintended effect of worrying students too much about plagiarism accusations. A better explanation about how keystroke data is used may help alleviate student anxiety, and is important to anyone who uses keystroke data for plagiarism detection. We are also improving the playback feature – fixing bugs and making it more obvious to the user – to give students a better idea of what the instructor can see from their keystroke data. Instructors can also improve by giving students a better explanation of precisely what constitutes cheating, and informing them that they are given a chance to defend themselves in the event of an accusation – something the playback feature could aid in.

4.4 General Attitude

Student attitudes towards requiring keystrokes are generally positive, with 60% indicating it is a good idea, and only 18% disagreeing. See Figure 6. In the free response question we found 11% of responses explicitly stating that the plugin was a positive thing, and 6% explicitly stating that it was a negative thing. Students who did not plagiarize responded positively, expressing that requiring keystroke data made the class more fair: "yes, very good. i love anything that evens the playing field and makes people more accountable."

> I completely forgot it was there since I was never influenced to plagiarize anything, but I feel like it is a very useful tool to catch those who are trying to plagiarize.

We also found 1.6% of students felt requiring keystroke data showed a lack of trust towards students, but still accepted it: "it shows a lack of trust in your students but other than that it's fine."

Privacy was a concern for some students. In the free response answers to our survey, 6% of students mentioned that the plugin felt invasive.

> I did not like the plugin at all. I would have never voluntary downloaded a key logger to my computer no matter what its intended purpose was if it wasn't a requirement for any of my assignments to get graded.

A benefit to using a plugin for student IDEs is that students remember it is logging their actions. 58% of students indicated that they always remember it is installed in the survey. They know if they cheat, their instructor could see what they did. In the free response 6% of the students said they did not forgot about the plugin, and 11% said they did forget about the plugin.

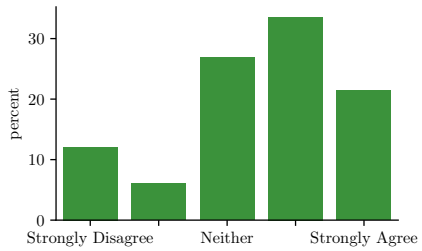

Figure 6: Student responses to "I think the plugin was a good idea".

5 CONCLUSIONS

Many methods have been developed to detect plagiarism in computer science, but without proper deterrence, they only find the problem after it has occurred. By simply collecting student keystroke data we found evidence of plagiarism deterrence. Student attitudes were mostly positive as they appreciated that it evened the playing field with their peers, but we also found evidence that anxiety among honest students may increase if they don't understand how detection methods work.

Keystroke data can accomplish plagiarism detection and help identify false positives from file similarity. Keystroke data is a minimally invasive method to record student effort on programming assignments and deter plagiarism. Only 6.4% of our students indicated that requiring keystroke data felt invasive. Computer science classes are typically focused on teaching the process of solving problems with computer science, but are typically graded on the final solution. Some courses have adopted the use of version control systems to record student effort. Keystroke data works similarly to version control systems, but at a higher resolution. We found in our survey that requiring students to submit keystroke data with their assignments helped them follow the proper programming process, and discouraged them from plagiarizing.

In this paper we have presented our experience collecting keystroke data to detect and deter plagiarism. We have discussed how keystroke data can better detect plagiarism, increase the difficulty to obfuscate plagiarism, and detect false positives from file similarity. We have shown that requiring students to submit keystroke data is a minimal invasion of privacy that deters students from plagiarizing, and can prevent innocent students from plagiarism accusations. Students were given a survey to express how they felt towards these topics and the requirement of keystroke data. Sixty percent of our students responded in support of keystroke data requirements, but 9.5% expressed anxiety caused by the requirement. To deter plagiarism and limit student anxiety, instructors who require keystroke data should explain clearly to students what plagiarism is and how keystroke data can detect both plagiarism and false positives.

References

[1] Alireza Ahadi, Raymond Lister, and Arto Vihavainen. 2016. On the number of attempts students made on some online programming exercises during semester and their subsequent performance on final exam questions. In *Proceedings of the 2016 ACM conference on innovation and technology in computer science education*. 218–223.

[2] Alireza Ahadi and Luke Mathieson. 2019. A comparison of three popular source code similarity tools for detecting student plagiarism. In *Proceedings of the Twenty-First Australasian Computing Education Conference*. 112–117.

[3] Mousumi Banerjee, Michelle Capozzoli, Laura McSweeney, and Debajyoti Sinha. 1999. Beyond kappa: A review of interrater agreement measures. *Canadian journal of statistics* 27, 1 (1999), 3–23.

[4] Brett A Becker. 2016. A new metric to quantify repeated compiler errors for novice programmers. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. 296–301.

[5] Mark Brimble and Peta Stevenson-Clarke. 2005. Perceptions of the prevalence and seriousness of academic dishonesty in Australian universities. *The Australian Educational Researcher* 32, 3 (2005), 19–44.

[6] Adam S Carter, Christopher D Hundhausen, and Olusola Adesope. 2015. The normalized programming state model: Predicting student performance in computing courses based on programming behavior. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. 141–150.

[7] Baojiang Cui, Jiansong Li, Tao Guo, Jianxin Wang, and Ding Ma. 2010. Code Comparison System based on Abstract Syntax Tree. In *2010 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT)*. 668–673. https://doi.org/10.1109/ICBNMT.2010.5705174

[8] Charlie Daly and Jane Horgan. 2005. Patterns of plagiarism. *ACM SIGCSE Bulletin* 37, 1 (2005), 383–387.

[9] Han De Vries, Marc N Elliott, David E Kanouse, and Stephanie S Teleki. 2008. Using pooled kappa to summarize interrater agreement across many items. *Field methods* 20, 3 (2008), 272–282.

[10] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, et al. 2015. Educational data mining and learning analytics in programming: Literature review and case studies. *Proceedings of the 2015 ITiCSE on Working Group Reports* (2015), 41–63.

[11] Matthew C Jadud. 2006. Methods and tools for exploring novice compilation behaviour. In *Proceedings of the second international workshop on Computing education research*. ACM, 73–84.

[12] J Richard Landis and Gary G Koch. 1977. The measurement of observer agreement for categorical data. *biometrics* (1977), 159–174.

[13] Juho Leinonen, Krista Longi, Arto Klami, Alireza Ahadi, and Arto Vihavainen. 2016. Typing patterns and authentication in practical programming exams. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. 160–165.

[14] Juho Leinonen, Krista Longi, Arto Klami, and Arto Vihavainen. 2016. Automatic inference of programming performance and experience from typing patterns. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. 132–137.

[15] Krista Longi, Juho Leinonen, Henrik Nygren, Joni Salmi, Arto Klami, and Arto Vihavainen. 2015. Identification of programmers from typing patterns. In *Proceedings of the 15th Koli Calling conference on computing education research*. 60–67.

[16] Lannan Luo, Jiang Ming, Dinghao Wu, Peng Liu, and Sencun Zhu. 2017. Semantics-based obfuscation-resilient binary code similarity comparison with applications to software and algorithm plagiarism detection. *IEEE Transactions on Software Engineering* 43, 12 (2017), 1157–1177.

[17] Karen L Reid and Gregory V Wilson. 2005. Learning by doing: introducing version control as a way to manage student assignments. In *Proceedings of the 36th SIGCSE technical symposium on Computer science education*. 272–276.

[18] Eric Roberts. 2002. Strategies for promoting academic integrity in CS courses. In *32nd Annual Frontiers in Education*, Vol. 2. IEEE, F3G–F3G.

[19] J. Saldana. 2015. *The Coding Manual for Qualitative Researchers*. SAGE Publications, Los Angeles.

[20] Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. 2003. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 76–85.

[21] Michael Stepp and Beth Simon. 2010. Introductory computing students' conceptions of illegal student-student collaboration. In *Proceedings of the 41st ACM technical symposium on Computer science education*. 295–299.

[22] Efrat Vinker and Amir Rubinstein. 2022. Mining Code Submissions to Elucidate Disengagement in a Computer Science MOOC. In *LAK22: 12th International Learning Analytics and Knowledge Conference*. 142–151.

[23] Christopher Watson, Frederick WB Li, and Jamie L Godwin. 2013. Predicting performance in an introductory programming course by logging and analyzing student programming behavior. In *2013 IEEE 13th international conference on advanced learning technologies*. IEEE, 319–323.