7th International Conference on Higher Education Advances (HEAd'21)
Universitat Politècnica de València, València, 2021
DOI: http://dx.doi.org/10.4995/HEAd21.2021.13072

Style features in the programming process which can help indicate plagiarism

Heidi Meier, Marina Lepp
Institute of Computer Science, University of Tartu, Estonia.

Abstract

In the new situation, where more and more final programming assignments are performed outside the classroom, it is necessary to pay more attention to the possibilities of understanding whether a student has created the solution on their own. To do this, it is possible to use a programming environment that logs user actions. One such environment is Thonny, which also allows the programming process to be replayed. The aim of this study is to identify style features of different learners, based on solution logs of introductory programming courses, and to explore how permanent these features are and can these indicate whether learners have solved the tasks without external aids. It can be said that non-programming style features, like the order of writing brackets or quotation marks, are more permanent and can be used to detect plagiarism. However, programming style features, such as the use of variable names or increment, are very variable between courses, and students participating in introductory courses do not have an established style. They are greatly influenced by the style features of teaching materials and solutions of sample tasks. Therefore, programming style features cannot be used to automatically check if a student has solved a task on their own.

Keywords: *Style features; programming process; plagiarism; higher education.*

1. Introduction

As time goes on, programming becomes more popular, and people increasingly learn it at universities. During the learning process, students solve many programming tasks. Knowing various aspects of the programming process is helpful for understanding how different students study. In the new situation, with the increasing share of e-learning, it is increasingly the case that even final assignments are performed outside the classroom. More attention needs to be paid to the ways of understanding whether programs have been created by students themselves. It is possible to use different recording tools. One option is to use a programming environment which logs user actions and use the ability to examine logs. One such environment is Thonny, a Python programming environment designed for learning and teaching programming, and it has logging functionality. The logs contain information about user actions during the solution process, and the programming process can be replayed (Annamaa, 2015). Because of logging details, it is also possible to identify from the log the order in which characters were written.

In our study, we analysed Thonny logs collected from the courses "Introduction to Programming" and "Introduction to Programming II" which took place at the University of Tartu in the spring of 2020. The study is qualitative and looks at style features that can be distinguished based on the programming process. The focus is also on the persistence of style features because permanent ones can be used in detecting plagiarism. The detected style features can later be used in quantitative research. The article is based on the following research questions. 1) What style features of different learners in solving programming tasks can be identified based on task solution logs of introductory courses? 2) What style features are permanent and can indicate whether a learner has solved the tasks on their own?

2. Literature Review

The similarity between programs has been studied to a great extent. A number of tools has been created, and some of them are also based on programming style. For example, Arabyarmohamady, Moradi and Asadpour (2012) developed a tool focused on the similarity of programs, in which they looked at the programming style and used data collected from professional programmers and first-year students. The system performed better than other systems in detecting whether code was copied from the internet or received from somebody outside the course (Arabyarmohamady *et al*., 2012). Ganguly, Jones, Ramirez-de-la-Cruz, Ramirez-de-la-Rosa and Villatoro-Tello (2018) also analysed the coding style and took it into account. They used a set of features where they distinguished lexical, structural and stylistic features. Stylistic features were, for example, the number of lines of code, the number of white spaces, the number of tabulations, the number of empty lines, the number of defined functions, average word length, the number of uppercase letters, the number of lowercase

letters, the number of underscores (Ganguly *et al*., 2018). However, there are fewer studies that use information about the programming process. Some of them use the recording of keystrokes. Byun, Park and Oh (2020) analysed the keystroke data collected from the students in an introductory programming course. They showed that using common n-graphs (n consecutive characters while typing) is an effective way to detect plagiarism. They developed a system that observed all pressing and releasing actions in the author's activity at the millisecond level, and they used press-flight time, release-flight time, dwell time, and break time while analysing keystroke dynamics for detecting plagiarism.

Longi *et al*. (2015) distinguish students based on the average time it takes to type digraphs in programming. They analysed data from an introductory course of programming, which lasted for seven weeks, and they used data from all study weeks. Their results indicated that there was potential in using digraphs for identifying students. They compared the average time it takes for a student to type any character, to type a specific character, and to type a specific digraph. The average time of typing specific digraphs was the most accurate indicator for identifying students (Longi *et al*., 2015). Leinonen, Longi, Klami, and Vihavainen (2016) developed a methodology to automatically distinguish novice programmers from those who are experienced. Their results showed that students' programming experience can be identified using keystroke data (Leinonen *et al*., 2016). Some digraphs are common in programming and rarely occur in natural languages (Leinonen, 2019). More experienced programmers type these digital graphs faster than average. For example, such a digraph is "i+" in Java (Leinonen, 2019). There is also an analysis of plagiarism behaviour in introductory programming courses with take-home exams, and special software was used to record the programming process (Hellas, Leinonen, & Ihantola, 2017). Afterwards, the researchers interviewed the students suspected of plagiarism and developed a typology of plagiarism on the basis of these interviews. They also found patterns that can be helpful in identifying students who have plagiarised, for example, a linear solution process and pasting parts of the solution. It is also possible to detect collaboration through alignment of the programming process but it does not detect students who received help from someone outside the course (Hellas *et al*., 2017).

Schneider, Bernstein, vom Brocke, Damevski, and Shepherd (2018) developed a mechanism which compares program creation processes. They used logs containing sequences of events that were collected automatically during the programming process. Detection is based on comparing the histograms of command use in the logs. In plagiarised works, the log is too different from "honestly created" logs or too similar to another log (Schneider *et al*., 2018). Blikstein (2011) developed a technique based on hundreds of snapshots to analyse and categorise students by programming experience. He used logs that contained all users' actions, for example, keystrokes and changes in the code. He studied what strategies students use to solve a programming task, for example, using an existing program as a starting point,

taking breaks in coding while browsing other sample programs or thinking of solutions, linear growth in the code size, trial-and-error strategy, sudden increase in code size due to pasted code (Blikstein, 2011).

3. Methodology

Data were collected from the courses “Introduction to Programming” and “Introduction to Programming II” which took place at the University of Tartu in the spring of 2020. These were elective courses for students who were not studying computer science as a major. The course “Introduction to Programming” lasted for seven weeks, the main topics included: variables, conditional statement, loop, list, reading from a fail, writing to a fail, function, simple user interface. Those students who completed the course "Introduction to Programming" could, if desired, continue with the course "Introduction to Programming II". The course “Introduction to Programming II” lasted for six weeks and covered the following topics: nested loops, dictionaries, tuples, sets, graphics and recursion. Both courses were organised in Python. At the beginning of the course “Introduction to Programming”, there were both lectures and practical sessions in the classroom. Then, however, an emergency began due to the pandemic, and the course was completely transformed into an e-learning format. “Introduction to Programming II” was entirely in the e-learning form. The final assignments also took place outside the classroom.

The first course had 140 participants, 118 of whom completed it. 39 students continued with "Introduction to Programming II" and 22 of them completed it. In addition to homework, students had to submit a Thonny log file each time. They also had to add a Thonny log file when they submitted the solution to the final assignment. Students who met the following conditions were included in the study: 1) Studied in both courses; 2) Submitted final assignment programs and logs for both courses; 3) Submitted homework logs for both courses for at least 50% of the weeks. There were 17 such students.

All logs submitted during the two courses by the 17 students were analysed using Thonny’s functionality of replaying the programming process. In addition, the submitted programs were reviewed. Before the analysis, a table was compiled for each week with characteristics that could differentiate students. Information about each student was added to the table during the log analysis. If potential new style features were noticed during the analysis, these were added to the tables. When all logs were analysed, the information collected from each student's logs for different weeks was compared. Also, the materials used by the students in the study process were analysed. In particular, the extent to which the students' style features overlapped with those in the study materials was compared. The style features in students' homework were compared with those from the corresponding chapter of the study material.

4. Results

The style features that were focused on in the study can be divided into two types: 1) non-programming style features; 2) programming style features. Non-programming style features include the order of writing brackets, quotation marks, apostrophes, and square brackets. For example, they can start by writing an opening bracket, then text and finally the closing bracket (in Table 1 (x) "x" [x]) or an opening bracket, the closing bracket, and finally the text inside (in Table 1 () "" []). Based on the main writing order of brackets, quotation marks, apostrophes, and square brackets, the students were divided into seven types (Table 1).

Table 1. Writing parentheses, quotation marks, apostrophes, and square brackets.

Type number	Type	Number of Students
1	(x) "x" [x]	8
2	() "" []	4
3	(x) "x" []	1
4	() "x" []	1
5	(x) 'x' []	1
6	Mixed: (x) "x" [x] and () "" []	1
7	Mixed: (x) 'x' [x] and () " []	1

Students' style of writing brackets, etc., was the same from week to week and did not change significantly during either course. The order of writing is intuitive, and they probably do not think much about it. It was also examined whether students use quotation marks or apostrophes while programming. 15 out of 17 used mainly quotation marks. There were also quotation marks in the study materials. Generally speaking, the basic style of writing quotation marks or apostrophes was the same from week to week. Some students used apostrophes in specific contexts, even when their main style was to use quotation marks. One student, who usually used apostrophes, began writing in quotation marks while writing the final assignment and later corrected them into apostrophes. It could be an indication of plagiarism.

Also, it was examined whether or not the students used spaces in expressions. 1 out of 17 wrote without spaces, 2 used a mixed style, and 14 wrote mainly with spaces. However, while the use of spaces is not persistent, the student who systematically did not use spaces continued to write without spaces during the two courses. One student sometimes added a space between the function name and the following opening bracket. She did it somewhere every week, including in the final assignment of both courses. Also, it was noticed that study materials influence the use of spaces. Generally, spaces are used in study materials. There

are some examples in some chapters, where there are no spaces somewhere, and the same use of spaces was found in students' solutions.

In the following, the features that are more related to programming are analysed. For example, the writing of a print statement was considered. Most students used concatenation of strings in the print statement; only some of them used commas. In the course "Introduction to Programming", the results were as follows (including only the weeks with tasks using print statements): week 3 – student 2, 4, 6, 8; week 4 – student 4, 8; week 5 – student 4, 6, 8; week 6 – student 2, 4, 6; final assignment – student 4, 6, 8, 12. In the course "Introduction to Programming II", only some used commas: week 1 – student 4, 8, 12, 17; final assignment – student 4, 6, 8. Most students who used the alternative variant did so several times, but not every time. In the study materials, there is a print statement with concatenation of strings. One student previously preferred joining the strings, but used commas in the final assignment. When the logs were analysed, it was clear that he had a solution of a sample task with commas open during the programming. Based on this example, we can see that sample solutions also affect the style.

The use of variable names was also considered. Most students wrote multi-word variable names with a lowercase letter or used an underscore. The following students used capital letters in the middle of variable names: week 3 – student 5; week 4 – student 5; week 5 – student 5, 6, 9, week 6 – student 1, 3, 6, 9 (the 5th student did not upload logs and programs); final assignment – student 5, 9. In the course "Introduction to Programming II", only some used capital letters in the middle of variable names: week 2 – student 5; week 3 – student 8; final assignment – student 9. In the other weeks, no one used the alternative variant. Students varied the variants and using variable names was not persistent. Variable names are separated by underscores in the materials or several words are written together. In the materials of week 6, it is different: there, capital letters are used in the middle of a word in variable names. Students used the second style the most in week 6.

Next, the use of i += 1 vs i = i + 1 was studied. Students used both options. It was often the case that they tried one of these at first and then the other one. More use of i += 1 was observed towards the end of the course and during the second course (Table 2). The topic for the third week was a loop, and they used it then for the first time. The variant i = i + 1 is used at the beginning of the 3rd-week materials and then i += 1 is used in the following subsections. The 4th-week materials have i += 1, and the 5th-week and 6th-week materials also have i += 1. Comparing the student's choices with the materials, it can be said that the materials significantly influenced what choices students made in their programs.

The writing of the module import was also analysed. Most of the students used the variant shown in the examples in materials. For example, most students used the 'from random import *' style to import the 'random' module, not 'import random'. In week 3, students 6, 11, 17

used the alternative variant; nobody did it in week 6. In the second course, only the 2nd student used the alternative variant. Importing modules was only included in a few tasks. Therefore, no conclusions can be drawn about the permanence of module import writing.

Table 2. Use of i = i + 1 and i += 1.

Introduction to Programming			**Introduction to Programming II**		
Week	Style feature	Number of students	Week	Style feature	Number of students
Week 3	only i = i + 1	1	Week 1	only i = i + 1	0
	only i += 1	9		only i += 1	14
	both	7		both	3
Week 4	only i = i + 1	3	Week 2	only i = i + 1	2
	only i += 1	13		only i += 1	11
	both	1		both	1
Week 6	only i = i + 1	1		missing	3
	only i += 1	16	Final assignment	only i = i + 1	2
	both	0		only i += 1	15
Final assignment	only i = i + 1	1		both	0
	only i += 1	15			
	both	1			

To open the file, students mostly used the form 'file = open("data.txt", encoding = "UTF-8")'. The alternative option 'with open("data.txt") as file' was not used by anyone in week 4; in week 6 it was used by one student in one task. The same student used it throughout the second course, as well as in the final assignment. The form 'file = open("data.txt", encoding = "UTF-8")' is used in the study materials. The file opening form was also analysed in more detail (if the students added encoding or not or added, for example, ‘r’). It can be said that it varied from week to week, and the students did not do it the same way every time.

5. Conclusion

Based on the results, it can be said that non-programming style features are more permanent. They can be used to automatically check if a student has solved a task on their own. However, since some students belong to the same type, it is necessary to use various style features and combine them with, for example, the average time of typing digraphs, etc. The non-programming style features can also be combined with methods that check the similarity of the programs. However, the programming style features vary and change during the course. The students participating in introductory courses do not use programming style features in a persistent manner; it is significantly influenced by how these features are used in study materials. Final assignment solutions are also influenced by the solutions of sample exercises

that the students use while writing the program. Programming style features cannot be used to automatically check if a student has solved a task on their own.

It is important to further explore the style features with quantitative methods that help differentiate the students' programs and develop tools to help control possible plagiarism cases as, because of the current situation, there are more and more cases where students solve final assignment tasks at home and teachers cannot see who exactly solved the task. Finally, it should be noted that the main limitations of this study are the small sample and the length of the courses. It is possible that the results are not quite the same if the target group of the course is different or if it is not an introductory course.

References

Annamaa, A. (2015). Thonny: A Python IDE for Learning Programming. *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, 343. doi: 10.1145/2729094.2754849.

Arabyarmohamady, S., Moradi, H., & Asadpour, M. (2012). A Coding Style-based Plagiarism Detection. *2012 International Conference on Interactive Mobile and Computer Aided Learning (imcl)* (pp. 180–186). New York: IEEE.

Blikstein, P. (2011). Using learning analytics to assess students' behavior in open-ended programming tasks. *Proceedings of the 1st International Conference on Learning Analytics and Knowledge*, 110–116. doi: 10.1145/2090116.2090132.

Byun, J., Park, J., & Oh, A. (2020). Detecting Contract Cheaters in Online Programming Classes with Keystroke Dynamics. *Proceedings of the Seventh ACM Conference on Learning @ Scale*, 273–276. doi: 10.1145/3386527.3406726.

Ganguly, D., Jones, G. J. F., Ramirez-de-la-Cruz, A., Ramirez-de-la-Rosa, G., & Villatoro-Tello, E. (2018). Retrieving and classifying instances of source code plagiarism. *Information Retrieval Journal*, *21*(1), 1–23. doi: 10.1007/s10791-017-9313-y.

Hellas, A., Leinonen, J., & Ihantola, P. (2017). Plagiarism in Take-home Exams: Help-seeking, Collaboration, and Systematic Cheating. *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, 238–243. doi: 10.1145/3059009.3059065.

Leinonen, J. (2019). *Keystroke Data in Programming Courses*. Helsingin yliopisto.

Leinonen, J., Longi, K., Klami, A., & Vihavainen, A. (2016). Automatic Inference of Programming Performance and Experience from Typing Patterns. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 132–137. doi: 10.1145/2839509.2844612.

Longi, K., Leinonen, J., Nygren, H., Salmi, J., Klami, A., & Vihavainen, A. (2015). Identification of programmers from typing patterns. *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, 60–67. doi: 10.1145/2828959.2828960.

Schneider, J., Bernstein, A., vom Brocke, J., Damevski, K., & Shepherd, D. C. (2018). Detecting Plagiarism Based on the Creation Process. *IEEE Transactions on Learning Technologies*, *11*(3), 348–361. doi: 10.1109/TLT.2017.2720171.