Developing students' code style skills

Yuliia Prokop
Department of Computer Science
Czech Technical University in Prague
Prague, Czech Republic
https://orcid.org/0000-0002-6608-3668

Olena Trofymenko
Department of Information Technologies
National University "Odessa Academy of Law"
Odesa, Ukraine
https://orcid.org/0000-0001-7626-0886

Olexander Zadereyko
Department of Information Technologies
National University "Odessa Academy of Law"
Odesa, Ukraine
https://orcid.org/0000-0003-0497-9861

2023 IEEE 18th International Conference on Computer Science and Information Technologies (CSIT) | 979-8-3503-6046-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/CSIT61576.2023.10324182

***Abstract* — Developing the skills of writing clean code in novice programmers is an urgent and essential task for today's university teachers. Despite numerous studies of this problem, the optimal solution has not yet been found. The research examines the motivation of students to adhere to the code style without any particular methodological influence. It is shown that students with an average of 2.2 years of programming experience understand code style rules and say they are ready to follow them. However, their code contains numerous style errors, which indicate the need for additional pedagogical efforts. In the second part of the study, we present the results of the experiment with another group of students. We applied the method of penalizing students for violating code style requirements, and it showed a positive impact: the total number of errors decreased. Optimistic results were obtained for many criteria. However, the same approach should be applied in subsequent courses to consolidate a positive impact. The paper proposes a method for developing students' code style skills.**

Keywords — code style, code quality, programming, education

I. Introduction

In practice, a software product is often developed by some programmers and improved and maintained by others. Maintenance is the most time-consuming stage of the software product's life. The quality of the initial code greatly influences the maintenance costs. Low-quality programs may cause severe problems in software systems. That is why code quality is one of the most critical competencies of a future programmer. It includes various aspects, such as code style, readability, maintainability, complexity, etc.

The knowledge and skills to write high-quality code mostly come with experience. Frequently, students learn the syntax of programming languages, develop fundamental programming skills, and then relearn how to write easy-to-read code that meets specific style requirements. This approach is ineffective because research [1] shows that coding style skills are more easily mastered by students who are just starting to learn programming than those who already have three or more years of programming experience. If students are not guided to write "good" code early, this might result in additional time and effort students have to invest in later classes [2]. This is why teaching the coding style from the first year when learning the basics of programming is relevant.

Identifying common code quality mistakes, particularly code style, and developing methods to teach students how to write clean code are essential tasks for today's programming teachers.

This paper aims to explore students' motivation and understanding of code style rules in the absence of a particular pedagogical influence. The study also investigates the improvement of code style in student programs when using specific exposure: penalizing for requirement violations and providing feedback.

II. Literature Review

Teachers worldwide have actively discussed the problem of developing clean code-writing skills for IT students during the last decade.

The attention of researchers is directed, in particular, on revealing typical code quality mistakes students make. The rules of code writing style are not universal and depend on a programming language. That's why common mistakes can be different in different languages.

Numerous papers are devoted to code quality errors in programs in specific programming languages and development environments. For example, in [2], programs in Java written with BlueJ are studied. The authors of [3] have also revealed the Java code quality requirements, which students most often violate. The authors of [4] have analyzed 114 000 solutions of 161 short coding tasks and compiled a catalog of 32 code quality defects. They found that most correct programs contain flaws and that students do not stop making them if they do not receive targeted feedback.

The paper [5] summarizes typical code quality defects from seven programming courses with different levels of complexity in Java and Python. The study showed that in Java, students frequently neglect to add blank lines between code components and braces when they are optional. They sometimes use too many parentheses in expressions, too much code in one line, or too many variables in one declaration. In Python, students often neglect to put a space after the comment prefix and have too much text on one line.

The authors of [6] showed that formatting and documentation issues are the most common in student code. Other typical issues are error handling, methods structuring, modularization, and code documentation [2].

Researchers point out the difficulties in teaching students to write clean code. Many works are devoted to various techniques that help achieve positive results in this field. For example, one of the thesis chapters [7] is dedicated to teaching programming style in the CS1 course. The author proposes an alternative pedagogical approach in which students are given brief instructions on the proper style, and then students critique and correct examples containing the wrong style.

According to the authors of [9], the best results come from combining the three: a teacher explaining the rules and errors, an automatic AutoStyle check, and style evaluation with feedback. A study [7] has also shown that the best results in the learning process are obtained using the automatic coding style evaluation proposed by the author, which is accompanied by feedback.

The paper [4] describes code quality defect detectors that can be used to generate valuable feedback for students automatically.

The authors of [7] investigate the effectiveness of human style assessors and universal static analysis tools for evaluating specific style criteria. It is found that human evaluators do not provide reliable correspondence of style scores when compared to each other for several requirements. Furthermore, human assessors show inconsistent results on objective style assessment criteria compared to static analysis checks of the same rules. At the same time, the authors conclude that existing general-purpose static analysis tools are insufficient to evaluate all existing style criteria. In particular, static analysis tools cannot yet assess the names of variables and functions clearly and precisely. However, combined with an automated evaluation system, these tools can be used to reduce the efforts of human evaluators.

Unfortunately, teachers do not always pay enough attention to code quality in the first courses [10]. The work [11] investigates teaching results of 141 university courses on introduction to programming and found that only about 30% mention a topic related to code quality. The authors of [12] study how teachers pay attention to code quality in teaching, what typical mistakes they notice, and how they help newcomers improve their code. It is found that only a little more than half of teachers pay attention to code quality while teaching programming to first- and second-year students and consider it a significant aspect.

Thus, the problem of developing students' code style skills has attracted the attention of many modern researchers. But an optimal solution hasn't been found, so the question requires more research and pedagogical experiments. Besides, most papers investigate style rules in Java and Python programming languages. However, first-year students often learn C/C++ [13], which has some specific requirements for code quality.

III. CODE STYLE STUDENTS' KNOWLEDGE WITHOUT SPECIFIC PEDAGOGICAL EXPOSURE

As a rule, at the beginning of the first programming course and later during the lectures and practical classes, teachers draw students' attention to at least the basic rules of code style: naming variables, using indents and curly braces, program decomposition, striving to minimize the lifetime of a variable, etc.

But, as practice shows, students often approach these rules thoughtlessly and don't follow them because simple programs can be executed error-free, ignoring code style requirements. Teachers, on their part, don't demand compliance with the rules and are typically only restricted by oral comments. This leads to students knowing coding style rules only theoretically and not using them in programs.

To the teacher's question, "Why don't you use indents in the program?" a first-year student typically answers: "It works without indents". It seems to them that code formatting rules can be ignored in small programs. However, this habit is gradually carried over to more extensive programs.

Besides students' personal qualities, such as laziness, desire to minimize rules and restrictions, and reduced time spent on writing programs, there are also pedagogical reasons for students' low motivation to follow code style rules.

All the training examples teachers give students in lectures and tutorials must fully comply with all the style rules. However, teachers frequently ignore some code style requirements in their samples. And this significantly reduces students' motivation. Insufficient motivation and persistence among teachers are another possible reason [12].

To study students' motivation and identify their knowledge of code style, we surveyed novice programmers (first- – second-year students). Over 100 students of Odesa Polytechnic National University and National University "Odesa Law Academy" participated in the survey. Most of the participants are second-year software engineering students. Their experience in programming ranges from 0.5 years (minimum) to 8 years (maximum). The average experience is 2.2 years.

Answers to the question "*What programming languages do you use?*" were distributed as follows: Java – 58, Python – 56, C++ – 53, JavaScript – 36, C# – 28, C – 18, and others – 16.

To the question: "*Do you know what 'coding style' and 'clean code' mean?*" 82% of participants answered in the affirmative. 16.1% said they had heard something but were not sure of the meaning of these phrases. The rest (1.9%) gave a negative answer.

To the question: "*Is it necessary to follow all code style requirements?*" students gave the following answers:

- Yes, it is necessary to follow all requirements – 25%;
- Only basic rules should be followed; the rest are at the programmer's discretion – 58%;
- Not necessarily, but it is desirable; it is entirely at the programmer's discretion – 17%;
- No need to follow – 0%.

That is, 25% of our survey participants believe complying with all requirements and rules is necessary. These results are comparable to those obtained by H. Keuning. When she asked: "*What do you think about paying attention to code quality?*" the students who participated in that survey answered: 24% – very important, 67% – important, and 9% – neutral [14]. However, in our case, the percentage of those who are neutral (considering the code style requirements optional) was higher.

Of those participants in our survey who answered that it is mandatory to follow the rules, five students (20%) were not sure that they understood the concepts of "code style" and "clean code", and 3 participants could not name any code style requirements. So, they cannot implement their readiness to write clean code without additional instructions.

In the last question, the survey participants were asked to name the code style requirements they knew. No options were given; students had to formulate them on their own. There were no restrictions on the programming language.

We grouped the requirements from students' answers by content and sorted them by frequency of mention (Table 1). The most common rules were giving variables and function names that clarify their purpose, accompanying complex parts of the code with comments, and using unified indents. Other requirements have a much lower frequency of mention.

The analysis of the answers revealed that some of the students who indicated that they knew the term "code style" misunderstood this concept.

No statistical analysis of the survey participants' code was conducted. But a review of their programs revealed numerous violations of even basic requirements, such as the use of indentation, variable names, decomposition, etc.

TABLE I. TOP 10 REQUIREMENTS FOR THE CODE STYLE NAMED BY THE PARTICIPANTS OF THE SURVEY

Requirement	Frequency of mention
Give clear names to variables and functions	64
Add comments to complex parts of code	56
Use proper indentation	54
Insert a blank line to separate parts of the code	23
Use a specific style for variable names	16
Limit the length of a code line to a certain value	15
Avoid repetition of code blocks	13
Decompose a program into smaller subroutines	12
Use a specific arrangement of curly brackets	10
Insert spaces between binary operands	8

The survey results show that despite the lack of specific efforts from the teachers, most students are familiar with the concept of code style. However, they can mostly name a few of the basic requirements. In addition, they are not motivated enough and often do not follow the rules in their programs.

Thus, teachers should systematically and consistently apply specific methodological efforts to develop strong skills in writing clean code in the learning process.

IV. IMPROVEMENT OF THE CODE STYLE WITH PENALTIES METHOD

Consider now what can be achieved by emphasizing the code style and encouraging students to fulfill its requirements in the learning process.

The homework of 70 students in the C Programming course was analyzed. The participants were first-year cybernetics and robotics students at the Faculty of Electrical Engineering, Czech Technical University in Prague. Before that, they had spent one semester learning Python programming.

Professor Jan Faigl is the author and creator of this C Programming course. He also initiated a method of teaching the students the correct code style. During the semester, the students do five pieces of homework, four of which are assessed for style besides the code itself [15]. If they violate the rules, they get penalty points and can lose up to 100% of their coding points. The code style requirements follow the coding standards [16, 17]. Over the years, the course teachers' team has collectively adjusted the requirements based on the evaluation experience to improve the guidance of the students.

The rules to follow are specified for the C programming language. They are pretty relaxed and flexible. This refers to the style of naming variables, the use of indentation (tabs or a certain number of spaces), and the positioning of opening curly braces. The most important thing is the unified use of names, brackets, and margins. In addition, there was no requirement to insert a blank line to separate code sections visually and to put a space between the comment symbol and the text.

Code functionality is tested and evaluated automatically. Teachers check code style compliance in manual mode. The evaluation is accompanied by individual commentary for each student describing the failures. In addition, the most common mistakes are discussed during the next practical class.

The errors most often made by students partially coincide with those identified by other authors. At the same time, they have specific features related to C programming. For example, students often do not control the allocation or reallocation of dynamic memory.

Table 2 shows the results of identifying the most common violations of code style rules in four homework assignments (HW1, HW2, etc.). The table lists the requirements for the student code and the number of errors for each piece of homework. A rule violation in a program was counted only once, regardless of the number of such errors.

TABLE II. THE NUMBER OF CODE STYLE VIOLATIONS IN STUDENTS' HOMEWORK

Requirement	HW1	HW2	HW3	HW4
The meaning of a variable/function is described by its name, and format of the names is consistent within the program	10	4	6	0
A program line has a maximum of 120 characters	10	5	0	4
Indentation and use of spaces are unified within the program	15	10	0	0
The use of curly braces is unified within the program	8	1	1	1
Comments accompany more complex parts of the text	25	15	4	9
Macros are defined for non-trivial numeric literals	27	17	34	12
The program is divided into short and simple functions	30	10	16	10
Low nesting of conditions (IF) and loops (FOR/WHILE) is preferred	12	3	1	0
The program does not contain similar blocks of code	13	3	11	10
The use of global variables is avoided	3	15	1	1
Variables are introduced as close to the application as possible	0	10	7	2
Variable lifetimes are as limited as possible	0	13	0	1
Dynamic memory allocation is controlled	0	8	29	15

Table 2 shows that as the difficulty of homework increases, students get rid of some errors but make others. For example, the problem with passing array as a parameter between functions, which students try to replace using global variables, is only acute in the second homework. The requirement "Dynamic memory allocation is controlled" does not apply to the first homework, as it does not use dynamic memory allocation. In HW2, it is used minimally, and in HW3, students have the most problems with memory allocation, which can be seen in the increasing number of errors.

An overall assessment of the number of violations shows that the number of errors gradually decreases (Fig. 1).

Fig. 1. Total number of rules violations in students' programs

Some criteria, such as using unified indentation and curly braces, providing comments for complex code, avoiding

global variables, and declaring variables as close to where they are used as possible, improve performance noticeably. Others (use of the same code blocks, program decomposition) are more pessimistic.

In the last work of the semester, no code style control was provided. The number of violations, such as using long lines or exceeding the permissible level of nesting, has increased again. That means that code style skills have been developed but not fixed. To achieve a consistent result, constant monitoring is needed, including in subsequent courses.

V. A METHOD FOR THE DEVELOPMENT OF STUDENTS' CODE STYLE SKILLS

Analysis of successful cases described in contemporary research and experiments conducted by the authors allowed us to propose a method for successfully developing students' code style skills.

1. All code style requirements should be clearly stated, explained, and shown to students with positive and negative examples.
2. All examples the teachers use to present the course material should strictly follow all code style requirements.
3. Most homework must be evaluated from the program functionality and code style perspective. The style may be assessed as a penalty as a percentage of the maximum score for the work. These penalties should be sufficiently tangible to encourage students to avoid violating the style rules.
4. The student should receive feedback after each work, describing their mistakes. Providing an opportunity to correct violations of code style requirements, at least in the first works, and cancel penalties makes sense.
5. Code style requirements should be coordinated with teachers of other disciplines.

VI. CONCLUSIONS

The problem of teaching code style rules to undergraduates is topical, and no optimal solution has been found so far. The study shows that without special pedagogical efforts, students understand at least the basic requirements for code style and declare their readiness to follow them. However, their programs contain numerous code style errors. Further instructions from the teacher and efforts to motivate students are needed.

The paper also presents the results of using the method of penalties for violation of coding style rules. It has been demonstrated that a consistent approach to ensuring compliance, coupled with a reduction in points for violations in homework and the provision of feedback, leads to a decrease in code style errors. However, to consolidate and improve the results, it is necessary to continue code style control in subsequent courses.

Based on the findings and recommendations of teachers and the results of the code style penalty methodology, we propose a method for developing students' code style skills.

Further research could focus on improving the automation of code style checking and feedback generation to minimize teacher time and improve error detection.

REFERENCES

[1] S.Chren, M. Macák, B. Rossi, and B. Buhnova, “Evaluating code improvements in software quality course projects,” 26th International Conference on Evaluation and Assessment in Software Engineering, EASE'22, Association for Computing Machinery, New York, USA, 2022, pp. 160–169. DOI: 10.1145/3530019.3530036

[2] H. Keuning, B. Heeren, and J. Jeuring, “Code quality issues in student programs,” 22nd ACM Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE’2017, Bologna, Italy, 2017, pp. 110–115. DOI: 10.1145/3059009.3059061.

[3] C. Kohlbacher, M. Vierhauser, and I. Groher, “Common code quality issues of novice Java programmers: a comprehensive analysis of student assignments,” 15th International Conference on Computer Supported Education, 2023, pp. 349–356. DOI: 10.5220/0011715400003470.

[4] T. Effenberger and R. Pelánek, “Code quality defects across introductory programming topics,” 53rd ACM Technical Symposium on Computer Science Education, SIGCSE’2022, vol. 1, New York, USA, 2022, pp. 941–947. DOI: 10.1145/3478431.3499415.

[5] O. Karnalim and W. Chivers, “Work-in-progress: code quality issues of computing undergraduates,” IEEE Global Engineering Education Conference, EDUCON’2022, Tunis, Tunisia, 2022, pp. 1734–1736. DOI: 10.1109/EDUCON52537.2022.9766807.

[6] I. Albluwi and J. Salter, “Using static analysis tools for analyzing student behavior in an introductory programming course,” Jordanian Journal of Computers and Information Technology (JJCIT), 2022, vol. 6, pp. 215-233. DOI: 10.5455/jjcit.71-1584234700.

[7] A. Koehler, “A methodology for teaching from student errors in computer science education,” A Dissertation submitted in partial satisfaction of the requirements for the degree of Doctor of Philosophy in Computer Science by University of California, 2020, [Online]. Available: https://escholarship.org/content/qt9x472353/qt9x472353_noSplash_24e49bb14b8e5c530578c32c02186ddd.pdf.

[8] E. Wiese, M. Yen, A. Chen, L. Santos, and A. Fox, “Teaching students to recognize and implement good coding style,” 4th ACM Conference on Learning @ Scale, L@S'17, New York, NY, USA, 2017, pp. 41–50. DOI: 10.1145/3051457.3051469.

[9] J.S. Perretta, W. Weimer, and A. DeOrio, “Human vs. automated coding style grading in computing education,” ASEE Annual Conference & Exposition, 2019, pp.1–13. DOI: 10.18260/1-2--32906, https://peer.asee.org/32906.

[10] H.-M. Chen, W.-H. Chen, and C.-C. Lee, “An automated assessment system for analysis of coding convention violations in Java programming assignments,” Journal of Information Science and Engineering, 2018, vol. 34, pp. 1203–1221. DOI: 10.6688/JISE.201809_34(5).0006.

[11] D. Kirk, T. Crow, A. Luxton-Reilly, and E. Tempero, “On Assuring learning about code quality,” 22nd Australasian Computing Education Conference, 2020, pp. 86–94. DOI: 10.1145/3373165.3373175

[12] H. Keuning, B. Heeren, and J. Jeuring, “How teachers would help students to improve their code”, ACM Conference ITiCSE’19, Aberdeen, Scotland, UK, 2019, pp. 119–125. DOI: 10.1145/3304221.3319780.

[13] Yu. Prokop, E. Trofimenko, O. Zadereyko, N. Loginova, M. Gerganov, “Multivariate analysis when choosing the first programming language studied in universities”, IEEE UKRCON-2019, Lviv, Ukraine, 2019, pp. 1224-1228.

[14] H. Keuning, B. Heeren, and J. Jeuring, “Student refactoring behaviour in a programming tutor,” 20th Koli Calling International Conference on Computing Education Research, Koli Calling'20, 2020, pp. 1–10. DOI: 10.1145/3428029.3428043.

[15] B3B36PRG - Programování v C, [Online]. Available: https://cw.fel.cvut.cz/wiki/courses/b3b36prg/start.

[16] Linux kernel coding style, [Online]. Available: https://www.kernel.org/doc/html/v4.10/process/coding-style.html.

[17] 5 making the best use of C, [Online]. Available: https://www.gnu.org/prep/standards/html_node/Writing-C.html