Influence of Periodic Role Switching Intervals on Pair Programming Effectiveness

Bin Xu, Sheng Yan(✉), Kening Gao, Yu Zhang, and Ge Yu

School of Computer Science and Engineering, Northeastern University, Shenyang 110189, China
{xubin,gkn,zhangyu,yuge}@mail.neu.edu.cn, yansheng1117@foxmail.com

Abstract. Pair programming has been widely used in programming experiment teaching in programming courses. One of the important factors affecting the successful completion of pair programming is the timing of periodic role switching. We organized an experiment in the course of Python Programming for 102 freshmen who did not major in computer science. By comparing the accuracy of code submitted by students in the online judge system, we evaluated the influence of pairing programming on students' programming ability under three different periodic role switching intervals of 15 min, 20 min and 30 min, and collected students' perception towards pairing programming under different modes. We also made a standard to judge the normalization of code, to study the influence of pair programming on the normalization of code written by students. The results show that when the periodic role switching interval is 30 min, pairing programming is helpful for students to solve difficult problems, and it has a positive impact on the solution of subsequent problems after experiencing the process of solving difficult problems. When the periodic role switching interval is 20 min, students have a positive attitude towards pair programming. Therefore, the best switching interval can be set between 20 min and 30 min. However, in terms of code normalization, there is no significant relationship between the standard degree of student code and the switching interval of pair programming. We gave some explanations for this in this paper.

Keywords: Programming course · Pair programming · Assessment

1 Introduction

In the past few decades, computers have become a basic tool in people's daily life. Programming is also required by all walks of life as an important skill. Therefore, more and more people choose to study computer science courses in colleges and universities [1]. Pair programming has been proved that students' programming levels can be improved in programming courses [13,14,16]. In pair programming,

Supported by the National Natural Science Foundation of China (U1811261).

two students coordinate to build codes by playing different roles, the driver is responsible for writing codes, and the navigator is responsible for finding errors and providing feedback. Periodic role switching allows pairs of programming students to share their ideas and collaborate in writing higher-quality programs [5,17].

Previous studies have shown that there are many factors that have a significant impact on the effectiveness of pair programming [6,8,19], including whether students have been exposed to programming before and the differences between teachers' pair programming methods. We have studied the correct rate and standard degree of codes written by students at different switching intervals in pair programming to explore the influence of different times on the effectiveness of pair programming.

2 Background

Pair programming is a programming technique in which two programmers, usually from the same workstation, work together on a task. With the rise of extreme programming [2], it has been widely used in professional applications for the first time and has since become a common programming method in the software industry. Two programmers work side by side in front of the same computer. One enters the code, while the other reviews every line of code he enters. The person who enters the code is called the "driver" and the person who reviews the code is called the "navigator". Usually, two programmers switch roles regularly.

Pair programming, as a well-known learning programming strategy, has proved to be a more effective method than individual programming methods [7,12,20], so pair programming is also used as part of or in conjunction with various teaching methods [9,15]. One advantage of pair programming is accessibility. Pair programming enables students who have just come into contact with programming to have confidence in programming, which essentially lowers the threshold for students to learn to program [4,18].

Under the educational background, there have been many kinds of research on pair programming, such as exploring the correlation between self-reporting preferences [10] and curriculum performance [3] and the core contributors in projects [11]. Pair programming also enables students to constantly check their views on programming and exchange learning strategies. This expands students' learning programming strategies and helps to open up students' thinking mode. However, the factors that affect pair programming are still largely unexplored, so this paper makes a quantitative analysis of the periodic switching interval of the two roles in pair programming.

3 Method

3.1 Experimental Design

The data for this study come from the Python programming introductory course of the Northeastern University of China in autumn 2019. Students have seven weeks of computer programming assignments throughout the semester. We selected the last four weeks of the semester to conduct pairing programming experiments to ensure that students have a certain programming foundation. Students can choose whether to do pair programming or not, and the matching partners of pair programming are chosen voluntarily. In the experiment, our class time is usually 120 min, and we provide 15 min, 20 min, and 30 min for students to switch drivers and navigators. Students can choose the role switching interval of their group and the role switching interval chosen by each group of students also remains the same, that is, the students use the same role switching interval in the whole process of the experiment. There are eight programming problems in the assignments arranged in four weeks. All students do not adopt pair programming in the first week, and the students in the next three weeks will carry out group pair programming. In the whole experiment, once random matching partners are selected, the matching partners of each student are fixed. Figure 1 shows the design pattern of the experiment. We counted the behavior records of the code submission of students in the online judge system, and the homework was only rated as passed or failed. In the following chapters, we also give a standard method to evaluate the normalization of code written by students.

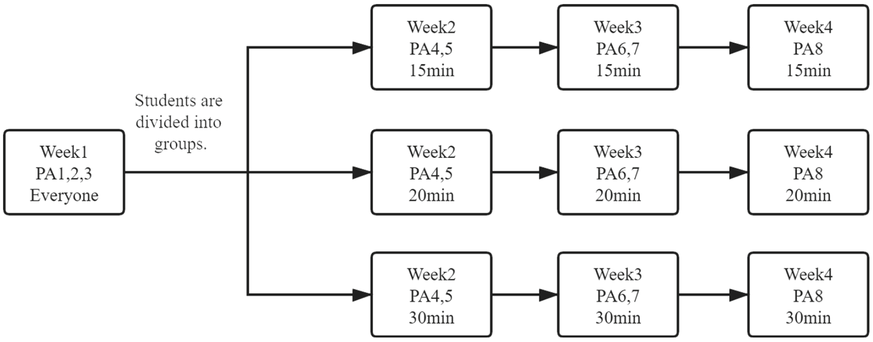

Fig. 1. Design of experiments for comparing three different switching times in pair programming.

3.2 Data

The data in this study were collected by two methods: (1) Homework records submitted by students in the online judge system; (2) Voluntary Pair Programming Strategy Questionnaire. The homework records submitted by the students in the online judge system include the time the students completed, the number of exercises completed, and whether they passed the evaluation. The questionnaire includes questions about pair programming activities. A total of 102 students participated in the survey, of which 96 students were grouped into pairs for programming, with a total inclusion rate of 94.1%. We selected 12 pairs of students from three groups respectively and took the remaining 12 students who did not adopt pair programming as the control group for experiments. The information we got from the questionnaire is shown in Table 1.

Table 1. Comparison of the number of people participating in the survey.

	15 min	20 min	30 min	Solo
Total number	24	24	24	12
Male	18	14	13	7
Female	6	10	11	5
Have programming experience	8	11	8	10
Heard of pair programming	8	9	4	6

In the experiment, each group of pair programming students completed an assignment together, and the students who programmed independently completed an assignment. We can see from Table 1 that some students have some programming experience. Therefore, before the experiment began, we conducted a test on the students to verify their programming level. We conducted two-sample T-tests on the scores of each group of students in different modes and the control group respectively, and the results are shown in Table 2.

Table 2. Differences in programming ability between groups.

Group	t	Degree of freedom (df)	p	Homogeneity of variance test (Sig)
15 min	−1.42	212.38	0.16	No
20 min	1.18	249.36	0.23	No
30 min	−1.13	253.00	0.26	No

The results showed that there was no significant statistical difference between the test results of each group and the control group. Therefore, the level of students' programming ability in our experiment is similar and will not affect the results of the experiment.

3.3 Research Questions

The research questions in this study are as follows.

– RQ1: What is the influence of different rotation intervals on the passing rate of students' pair programming code writing?
– RQ2: Will pair programming improve students' coding normalization?
– RQ3: What are the students' perceptions of pair programming under different switching intervals?

4 Result

4.1 RQ1: Code Pass Rate

To ensure that the students in each mode can accept the same amount of homework, we collected the passing rate of 8 program questions in the programming task and scored the passing rate of each question equally. The code passing rate of each homework in each mode can be calculated by the following formula:

$$R\left(G_i, P_j\right) = \frac{1}{N_i} \sum_{i}^{N_i} \frac{A\left(\theta_k^i, P_j\right)}{S\left(\theta_k^i, P_j\right)} \tag{1}$$

Where G_i represents the i-th mode; N_i represents the number of people in i-th mode; P_j represents the j-th homework; θ_k^i represents the k-th person in i-th mode; $A\left(\theta_k^i, P_j\right)$ represents the correct number of homework submissions in P_j; $S\left(\theta_k^i, P_j\right)$ represents the total number of homework submissions in P_j.

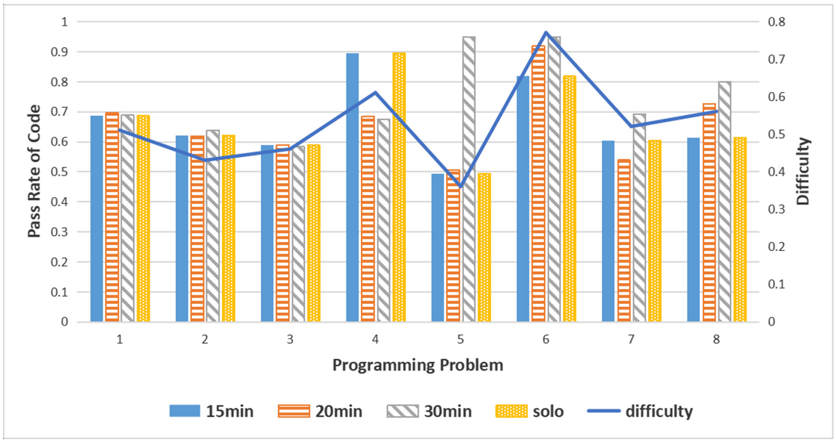

Fig. 2. Illustration of the passing rate of each group of codes and the difficulty of homework. (Color figure online)

According to the students' feedback and the weekly teaching content, combined with the code submitted by all the students, the change curve of homework difficulty can be obtained. As shown in Fig. 2, the yellow curve indicates the difficulty of the homework, and the smaller the ordinate indicates the greater the difficulty of the homework. The difficulty of homework is calculated from the records submitted by everyone on Online Judge, and the values of each point on the curve indicate the passing rate of each homework submission. The lower the pass rate, the more difficult the homework is, and the more difficult it is for the students to solve the homework. The cylindrical graph represents the passing rate of students' code submission in the online judge system under the switching interval of 15 min, 20 min, 30 min, and control group respectively. From the picture, we can find that the passing rate of students' codes is positively related to the difficulty of homework, which accords with our cognition.

In the phase of no pair programming, the three groups of students all adopt the single programming mode, and there is no obvious difference in the passing rate of homework submission. In the phase of pair programming, all the students begin pair programming. When solving the first homework, the group with a switching interval of 15 min has great advantages, can quickly solve some problems, and improve the passing rate of homework, while the other two groups have no obvious difference at the beginning. However, when the difficulty of homework increases, the group with a 30-min switching leads the other two groups by more than 40%.

Two-sample T-test was conducted with the pass rate data of codes with a switching interval of 15 min and 20 min and the data with a switching interval of 30 min respectively, and the difference between them is shown in Table 3. It can be seen that the group with a switching interval of 30 min has obvious statistical differences with the other two groups. Generally speaking, the difference in group scores is negative, which indicates that the other two groups are worse than the students whose switching interval is 30 min.

Table 3. Differences in performance between groups.

Group	t	Degree of freedom (df)	p	Homogeneity of variance test (Sig)
15 min & 30 min	−6.89	212.38	0.00	Yes
20 min & 30 min	−7.34	187.42	0.00	Yes

4.2 RQ2: Normalization of Codes

To determine whether pair programming can make students write codes more standardized, we have formulated a standard to grade students' codes with a standard degree, and detailed standard information is as follows:

1. Items with a weight of 3
   (a) Indentation: Use 4 spaces per indentation level. Continuation lines should align wrapped elements either vertically using Python's implicit line joining inside parentheses, brackets, and braces, or using a hanging indent.
   (b) Tabs or Spaces: Spaces are the preferred indentation method. Tabs should be used solely to remain consistent with code that is already indented with tabs.
   (c) Maximum Line Length: Limit all lines to a maximum of 79 characters.
   (d) Line Break: A-Line Break should before a Binary Operator.
   (e) Blank Lines: Surround top-level function and class definitions with two blank lines. Method definitions inside a class are surrounded by a single

blank line. Extra blank lines may be used (sparingly) to separate groups of related functions. Blank lines may be omitted between a bunch of related one-liners (e.g. a set of dummy implementations). Use blank lines in functions, sparingly, to indicate logical sections.

2. Items with a weight of 2
   (a) Source File Encoding: Code in the core Python distribution should always use UTF-8.
   (b) Imports: Imports should usually be on separate lines. Imports are always put at the top of the file, just after any module comments and docstrings, and before module globals and constants.
   (c) String Quotes: When a string contains single or double quote characters, however, use the other one to avoid backslashes in the string. It improves readability.
   (d) Whitespace in Expressions and Statements: Avoid extraneous whitespace.
3. Items with a weight of 1
   (a) Comments: Good code should have wonderful comments.
   (b) Naming Conventions: Naming in python code should conform to the naming specification.

The total score of the standard score is 30 points. If there is any nonconforming item in the code, the corresponding points will be deducted. Students have no professional foundation of programming before, so we pay more attention to the standardization of students' code layout. So when we calculate the deducted score, we give different weights to different items. For example, we give smaller weights to two items that enhance code readability (variable naming and code comments). The specific calculation method is as follows:

$$S = \beta \sum_{i=1}^{11} n I_i \tag{2}$$

Where I_i represent the item i in the table, β represents the weighting coefficient of each term, and the basic score of each item is 1 point.

We collected the codes of all students participating in pair programming and then calculated the standard rate according to the code length of each program. The standard rate of each group of modes takes the average value of the standard rate of all codes in that group. The formula of code normalization is calculated as follows:

$$N\left(G_i, P_j\right) = \frac{1}{N_i} \sum_{i}^{N_i} \frac{S_{sum} - S\left(\theta_k^i, P_j\right)}{L\left(\theta_k^i, P_j\right)} \tag{3}$$

Where $N\left(G_i, P_j\right)$ represents the standard degree of the code of the i-th mode, N_i represents the number of people in the i-th mode, P_j represents the j-th topic, S_{sum} represents the total score of each program, $S\left(\theta_k^i, P_j\right)$represents the score deducted by each program according to the standard items, and $L\left(\theta_k^i, P_j\right)$ represents the length of the code.

Figure 3 shows the changing trend of standard rates for each group of codes. It can be seen from the figure that there is no correlation between the standard

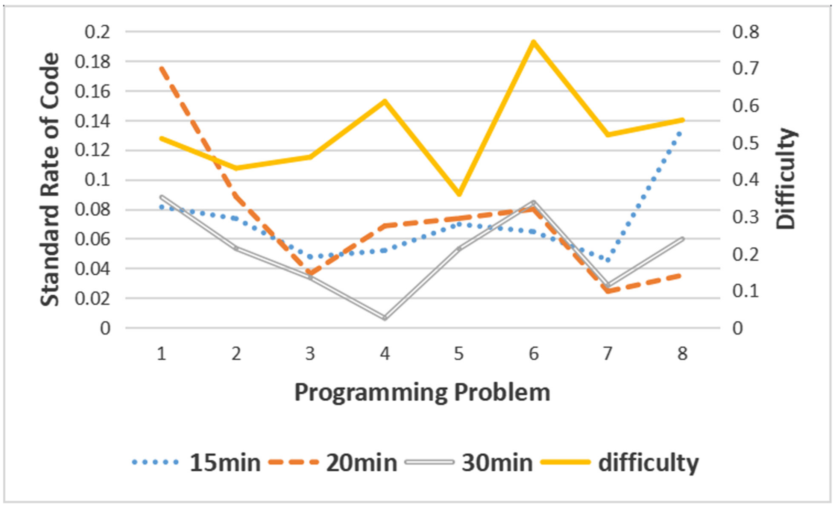

Fig. 3. A comparative diagram of the changing trend of the standard rate of codes in each group and the difficulty of homework.

rate and the difficulty of the homework, and the standard rate of each group has no obvious upward trend with the pair programming. We analyze that the reason why this may happen is that students learn and do while learning, which leads to the students' programming foundation not reaching the standard level. Therefore, we analyzed the codes written by students in the final examination and selected two questions to calculate the standard rate of codes written by students in the examination. The significance test of the regression coefficient for each group of data shows that there is no obvious linear correlation between the standard rate of codes and the time of pair programming.

4.3 RQ3: Students' Attitude to Pair Programming

After the experiment, to measure the students' experience in pair programming, a short questionnaire was distributed to the students.

- Q1: Are you willing to continue pair programming?
- Q2: Do you think pair programming is more efficient?
- Q3: Do you think pair programming is helpful to improve the programming level?

The results in Fig. 4 show that the students' attitude towards pair programming is positive. Most students are in favor of pair programming mode and are willing to continue pair programming. Figures show that students with a 20-min switching interval for pair programming feel more able to benefit from pair programming, and they feel that pair programming is more effective and can improve their programming level.

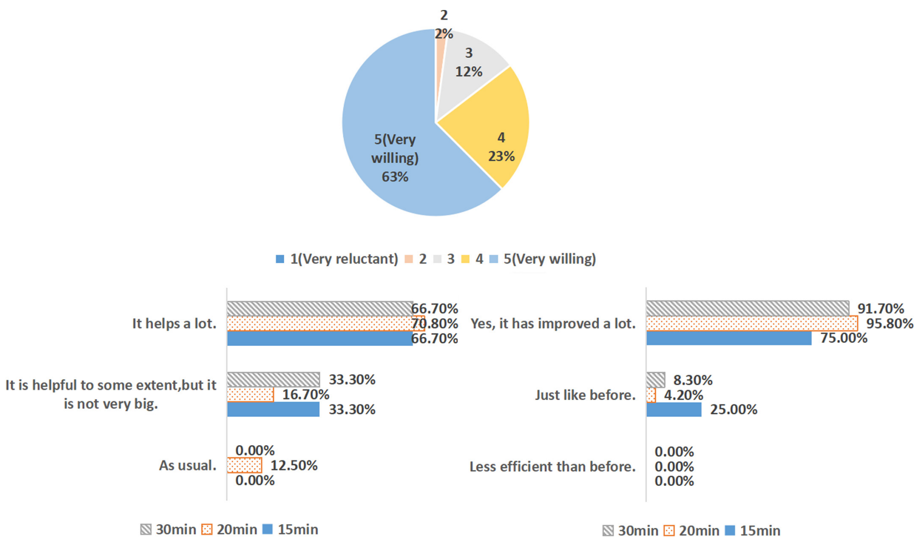

Fig. 4. Results of the questionnaire.

5 Discussion

The results show that the longer the pairing time is, the better it is for students to overcome difficult homework. Moreover, judging from the submission of homework after solving difficult problems, the passing rate of the submitted codes of this group of students is higher than that of the other two groups after experiencing cooperation in solving difficult problems. This shows that students benefit more from pair programming within 30 min.

In order to verify the validity of this result, we have a test for students after the pair programming course. It includes a simple programming problem and a programming problem beyond the syllabus. We counted the passing rate of each group of students. The results show that whether it is a simple question or a more difficult question, the students with a 30-min rotation time for pair programming have a higher pass rate. This shows that our previous results are valid.

As far as the normalization of codes is concerned, from the data obtained from the overall statistics, pairing programming has no effective influence. There is no significant linear correlation between the increase of pairing time and the standard degree of codes. We analyze the possible reasons for this situation include students' lack of programming experience, a small number of samples, and difficulty in homework. In order to verify whether it is related to the students' programming experience, we also selected the codes submitted by the students at the end of the final exam and calculated the standard scores of each group. The results show that after the end of the course, the code normalization has not been greatly improved, and there is no significant difference between the code normalization when the pair programming is not adopted and when the course is not studied. This reason is not the cause of this result. In the following research, we will expand the sample of the experiment and set up homework with different difficulties to analyze the normalization of the code.

In the process of practice, it is found that when the pairing programming mode is not carried out, the students with weak foundation are more dependent on the guidance teachers and treat the problems that cannot be solved passively. After developing the pairing programming mode, this part of the students can discuss it with the pairing members. According to the code submission time recorded by the system, it is found that these students can quickly solve the problems in programming, and students can obtain more satisfaction, and their learning enthusiasm and initiative have also been greatly improved. And we can see from the statistical results that students with a 20-min switching interval for pair programming think they can benefit more from pair programming. This also inspired us that the switching interval of pair programming is not as long as possible. Excessively long switching intervals will cause students to have a negative attitude towards pair programming.

6 Conclusion

In this study, we discussed the influence of periodic role switching interval on the effectiveness of pair programming and analyzed the pass rate, normalization, and students' cognitive attitude towards pair programming under different switching intervals. Our results show that students can benefit from pair programming when the switching interval for pair programming is 30 min. However, when we investigate students' attitudes, we find that students prefer to set the switching interval at 20 min.

Therefore, in the actual programming language teaching environment, we suggest that more attention should be paid to the pairing scheme of pairing programming. However, in the switching interval, we suggest to give priority to work continuity, and there is no need to specify specific time rigidly. According to the experimental results, to balance students' enthusiasm for pair programming and the benefits of pair programming in normal teaching, we suggest that the switching interval of pair programming in class can be set to 20–30 min.

When studying the influence of pair programming on the normalization of students' code writing, we got the opposite result. Our results show that there is no obvious linear correlation between the standard degree of code and the time of pair programming. We analyzed and verified that the reason for this result has nothing to do with whether the students study the whole course. In the following research, we will further expand the sample size and set up different difficult assignments to verify the remaining reasons.

References

1. Computer Research Association., et al.: Generation CS: computer science undergraduate enrollments surge since 2006 (2017). http://cra.org/data/Generation-CS/
2. Beck, K., Andres, C.: Extreme Programming Explained: Embrace Change. Addison-Wesley Professional, USA (2004). https://doi.org/10.5555/318762
3. Braught, G., Wahls, T., Eby, L.M.: The case for pair programming in the computer science classroom. ACM Trans. Comput. Educ. (TOCE) **11**(1), 1–21 (2011). https://doi.org/10.1145/1921607.1921609
4. Carver, J.C., Henderson, L., He, L., Hodges, J., Reese, D.: Increased retention of early computer science and software engineering students using pair programming. In: 20th Conference on Software Engineering Education & Training (CSEET 2007), pp. 115–122. IEEE (2007). https://doi.org/10.1109/CSEET.2007.29
5. Celepkolu, M., Boyer, K.E.: Thematic analysis of students' reflections on pair programming in cs1. In: Proceedings of the 49th ACM Technical Symposium on Computer Science Education, pp. 771–776 (2018). https://doi.org/10.1145/3159450.3159516
6. Chaparro, E.A., Yuksel, A., Romero, P., Bryant, S.: Factors affecting the perceived effectiveness of pair programming in higher education. In: Proceedings of PPIG, pp. 5–18 (2005)
7. Declue, T.: Pair programming and pair trading: effects on learning and motivation in a CS2 course. J. Comput. Sci. Coll. JCSC **18** (2003). https://doi.org/10.5555/771832.771843
8. Hanks, B.: Student attitudes toward pair programming. In: Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, pp. 113–117 (2006). https://doi.org/10.1145/1140123.1140156
9. Heinonen, K., Hirvikoski, K., Luukkainen, M., Vihavainen, A.: Learning agile software engineering practices using coding dojo. In: Proceedings of the 14th Annual ACM SIGITE Conference on Information Technology Education, pp. 97–102 (2013). https://doi.org/10.1145/2512276.2512306
10. Khan, S., Ray, L., Smith, A., Kongmunvattana, A.: A pair programming trial in the CS1 lab. In: Proceeding Annual International Conference on Computer Science Education: Innovation and Technology (CSEIT), pp. 6–7 (2010)
11. Liu, X., Bai, J., Liu, L., Ouyang, H., Zhou, H., Xu, L.: Mining core contributors in open-source projects. In: Ni, W., Wang, X., Song, W., Li, Y. (eds.) WISA 2019. LNCS, vol. 11817, pp. 690–703. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30952-7_70
12. McDowell, C., Hanks, B., Werner, L.: Experimenting with pair programming in the classroom. In: Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education, pp. 60–64 (2003). https://doi.org/10.1145/961511.961531
13. McDowell, C., Werner, L., Bullock, H., Fernald, J.: The effects of pair-programming on performance in an introductory programming course. In: Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education, pp. 38–42 (2002). https://doi.org/10.1145/563340.563353
14. Nagappan, N., et al.: Improving the CS1 experience with pair programming. vol. 35, pp. 359–362. ACM New York (2003). https://doi.org/10.1145/792548.612006
15. Porter, L., Simon, B.: Retaining nearly one-third more majors with a trio of instructional best practices in CS1. In: Proceeding of the 44th ACM Technical Symposium on Computer Science Education, pp. 165–170 (2013). https://doi.org/10.1145/2445196.2445248

16. Quintana, H., Grados, B.: Applying pair programming practice in the improvement of software design skills, in an undergraduate course. In: Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education, pp. 543–544 (2020). https://doi.org/10.1145/3341525.3393985
17. Reckinger, S., Hughes, B.: Strategies for implementing in-class, active, programming assessments: a multi-level model. In: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, pp. 454–460 (2020). https://doi.org/10.1145/3328778.3366850
18. Venkatesan, V., Sankar, A.: Adoption of pair programming in the academic environment with different degree of complexity in students perspective-an empirical study. Int. J. Eng. Sci. Technol. **2**(9), 4791–4800 (2010)
19. Xinogalos, S., Satratzemi, M., Chatzigeorgiou, A., Tsompanoudi, D.: Factors affecting students' performance in distributed pair programming. J. Educ. Comput. Res. **57**(2), 513–544 (2019). https://doi.org/10.1177/0735633117749432
20. Zhong, B., Wang, Q., Chen, J.: The impact of social factors on pair programming in a primary school. Comput. Hum. Behav. **64**, 423–431 (2016). https://doi.org/10.1016/j.chb.2016.07.017