Kashihara, A. et al. (Eds.) (2024). Proceedings of the 32nd International Conference on Computers in Education. Asia-Pacific Society for Computers in Education

Code Visualization System for Writing Better Code Through Trial and Error in Programming Learning: Classroom Implementation and Practice

Shintaro MAEDA[a*], Kento KOIKE[b] & Takahito TOMOTO[c]
[a]*Graduate School of Information and Computer Science, Chiba Institute of Technology, Japan*
[b]*Faculty of Engineering, Tokyo University of Science, Japan*
[c]*Faculty of Innovative Information Science, Chiba Institute Technology, Japan*
*front4.shintaro@gmail.com

Abstract: In programming education, refining code is crucial. Trial and error is one effective method for achieving this refinement. However, it can be challenging for learners to maintain motivation during conventional programming exercises due to them not knowing how good their code is and not being able to continuously perform trial and error. To address these issues, it is important to establish criteria for evaluating code quality and have learners objectively recognize them. Therefore, we propose and implement a method that uses a code visualization system that visualizes quality indicators, a ranking system, and a code viewing function based on that ranking. Evaluation of this system when implemented in a university classroom suggests that students write better code, based on these quality indicators, when they perform trial and error by themselves or by viewing other learners' code.

Keywords: Programming Learning, code Visualization, trial and error, motivation

1. Introduction

Trial and error, in which a learner attempts to improve pre-written code (hereafter referred to simply as "code"), is important when learning programming. This allows learners to enhance the performance and readability of their code, as well as other aspects that they may not have initially considered. Even after writing a code that meets the requirements, it is socially significant to aim for a better code, just as there is refactoring to replace the code with a more stable code (Keuning et al., 2021).

Motivating learners to write better code is not always easy. They will generally be satisfied with writing code that meets the given requirements, with little interest in engaging in further trial and error. One solution to this problem is to use code visualization systems. These systems produce a graphic representation of the learner's code, making it easier to understand visually (Sorva et al., 2013). However, we do not think that simply having learners use a code visualization system will necessarily make it easier for them to evaluate their own code, since these systems lack quality indicators (Task A) for evaluating better code and objective evaluation (Task B) for comparing their code to that of others.

We propose the following solutions to these problems. First, for Task A, we propose quality indicators by which the learner's code can be scored. Next, for Task B, we propose a ranking system based on these indicators that puts the quality of their code in context with that of others. Additionally, it is necessary to consider how learners are limited in their ability to write better code on their own. Specifically, since trial and error should ideally be a process of continual engagement, it is necessary to address the problem of not being able to continually write better code (Task C). To address Task C, we propose a ranking-based code browsing

function that encourages continual trial and error by allowing users to browse the codes of others as examples.

We aim to answer the following research questions in this paper: (RQA) what is the impact of trial and error on learners who use only the quality indicator and ranking system we propose, and (RQB) what is the impact of trial and error on learners who also use the ranking-based browsing control function? In RQA, the impact of trial and error is evaluated based only on the learner's ability, and in RQB, we evaluate the impact of trial and error by viewing other people's codes. We evaluated our system through practical application with second-year university students.

2. Related Work

2.1 Code Visualization Systems

Code visualization systems are tools that help learners understand program behavior by providing graphical representations of code execution (Sorva et al., 2013).

A well-known educational programming language that incorporates visualization is LOGO, which uses Turtle Graphics (Papert, 1980). Since this study emphasizes the importance of evaluation metrics, we focus on virtual robot programming, which is expected to support more diverse evaluation methods, and attempt to extend its capabilities.

2.2 Importance of Writing Better Code through Trial and Error

It has been pointed out how important refining one's code is when learning programming (Spinellis, 2003). Keuning et al. (2021) have described the importance of having learners better their code through refactoring, and they have developed a learning support system for this purpose. However, their system does not evaluate the learner's code specifically, since it presents pre-written code by the instructor as the problem. Programming is a creative task, and the importance of tinkering in bettering learners' code has been emphasized (Kotsopoulos et al., 2017).

2.3 Examples and Their Learning Effects

Examples that serve as models for learners are referred to as "worked examples" in educational psychology and other fields. Examples have been given for reducing learners' cognitive load and promoting efficient learning (Sweller, 2020). It has been shown that presenting examples promotes efficient learning for programming, but the effectiveness is limited (Zhi et al., 2019). Chen et al. (2020) proposed a method that adaptively determines how to present examples, emphasizing the importance of presenting appropriate examples.

Although there has been some debate about presenting proficient code to novice programmers, given concerns of the lack of learning due to level differences, Campbell and Bolker (2002) argued that having novice programmers read code written by experienced programmers has a positive learning effect.

2.4 Motivation and Effectiveness of Using Game Elements

Many studies have shown that integrating games into education enhances learner motivation. In this study, we adopt game-based learning, where educational content is delivered through game-like environments, to improve learner engagement and motivation (Campbell and Bolker, 2002). Specifically, we introduce quality indicators and rankings for code quality within a competitive game-based learning context, encouraging learners to compare their scores with others, thereby fostering a competitive spirit.

Research has demonstrated that incorporating competitive elements in game-based learning environments enhances critical thinking and motivation (Jiau et al., 2009).

Programming contests, a specific form of game-based learning, have become a popular method for improving programming skills by motivating learners through challenges (Paiva et al., 2020). Our study aims to leverage these competitive elements to encourage learners to continually improve their code quality through iterative practice and feedback.

3. Proposed System

Code visualization systems aid in understanding code by visually displaying its execution results. However, this alone may not be enough to improve code writing. The components of the system proposed in this study are listed below:

(***Quality Indicator***): Learners can be encouraged to write better code through trial and error by providing them with a means of evaluating code quality. It is important to evaluate the quality of code based on certain axes; code should ideally be highly productive, have low cost, and have high readability. As such, it is important that learners be shown in which categories their code is particularly strong. The quality indicator we propose in this study is scored according to a set of axes of evaluation. The different scores for specific axes of evaluation will help learners identify specific areas in which their code can be improved.

(***Ranking***): The quality indicator provides the learner with a score for their code on an evaluation axis. However, the score alone does not give learners a sense of how good their code is compared with that of others. The ranking system we are proposing provides them with this context by showing how their scores compare.

(***Viewing Control for Code Rankings***): Our ranking system includes a function that allows learners to view codes of a specific rank, thereby allowing them to organize other people's code based on quality. The added difficulty in replicating codes with higher rankings can dissuade learners from engaging in the continual trial and error we are trying to motivate, so it is important to control which codes the learner can view.

Accordingly, we developed a browsing control method that allows learners to view only codes that are close to their code's rank.

4. Case Study

(***Implementation of the Proposed System***): We implemented a code visualization system as a case study of the system discussed above. Specifically, the system is a crop harvesting game in which a robot executes a program causing it to move to a field, plant seeds, and harvest the crops that have grown after a certain time. All of this is visualized for the learner when the code is executed.

(***Implementation Quality Indicator***): The quality indicator in the case study uses the robot's behavior and the number of crops harvested as its axes of evaluation. Robot behavior incurs a cost, and the robot earns a number of harvests (productivity) when it harvests crops. A score is calculated for each indicator and visualized for the learner.

(***Ranking Learners' Codes***): The scores for the quality indicators for all learners using the system were tabulated, sorted, and posted. The learner's ranking for each quality indicator was displayed for them, showing them which of their codes were of a good quality and which required further trial and error. For example, if the productivity of their code is low, they can refer to the codes of others with high productivity to learn from them.

(***Control System for Rankings Shown***): The browsing control system set a constraint so that learners could only view codes that were one rank above their own and the codes of all ranks below theirs. The resulting incremental increase in difficulty better encourages engagement in the trial-and-error process. The restriction to viewing codes one rank above their own and all codes below their own is a provisional measure that takes into account the small data set of other people's codes.

5. System Interface

An example of the system interface is shown in the left portion of Figure 1. The learning task for this system is to program the robot to plant seeds in the field displayed in the center of the figure (the black panel indicates a field, and the white panel indicates a puddle) and to collect the crops that grow at that location after a certain time has elapsed. One unit of time (cost) elapses each time the robot executes a single command. There are multiple actions the learner is required to perform after planting a seed: moving to another field, planting a seed in that field, then recovering the crop in the field where the original seed was planted.

Three problems with certain characteristics were prepared for this system: Problem 1, a rectangular field; Problem 2, a mesh-like field with sections removed; and Problem 3, a square field with puddles.

An example of the ranking system for quality indicators is shown in the right portion of Figure 1. The ranking for a quality indicator is displayed in the center of the screen when it is selected, and the code associated with it is displayed on the right part of the screen. The system only allows learners to view codes that are below theirs in rank and at most one rank higher.

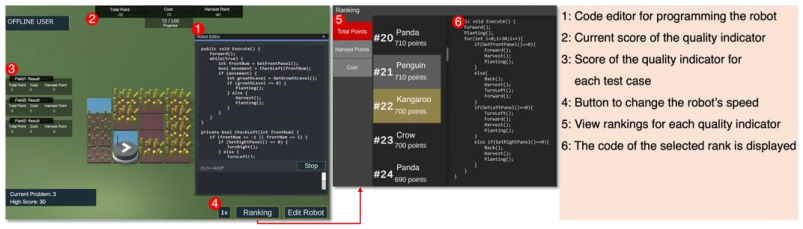

Figure 1. Example of the Code Visualization Interface and Ranking System.

6. Experiments

6.1 Overview

We conducted an experiment to evaluate the usefulness of the quality indicators, ranking system, and browsing control function used in the code visualization system.

Based on the RQs set up in Section 1, our hypotheses are that (H1) learners who use only the quality indicators and ranking system will write better code through trial and error based on the axes of the evaluation indicators (corresponding to RQ A), as will (H2) learners who use the viewing control function to view code rankings and (H3) learners who use the system's browsing control function with ranking (corresponding to B in RQ).

6.2 Outline of the Class Practice Session

The practice session was conducted in the following manner. Day 1 consisted of a system explanation (30 minutes) and a pre-test (30 minutes), and Day 2 consisted of learning to use the system (60 minutes), a post-test (30 minutes), and a questionnaire. Note that there was a one-week gap between the first and second days because they were undertaken during actual lectures. The pre-test and post-test were conducted using the system, which restricted the use of rankings. There were, however, no functional restrictions when teaching learners to use the system. The total number of participants was 65, but we excluded data from participants who did not agree to be assessed by the questionnaire and those who missed one of the lectures in the two-part experiment, so the final number of participants was 49.

6.3 Survey of Participants Using the Code Viewing Controls

The first step in the evaluation was to examine the number of participants who used the ranking-based browsing control function. The results show that many learners viewed others' codes in Problems 1 and 2, with 34 learners viewing the codes in both problems, while only 24 learners viewed the code in Problem 3. In contrast, the number of learners who did not view others' codes was 15 for Problems 1 and 2, and 25 for Problem 3. Problem 3 was more difficult than Problems 1 and 2 because of the puddle avoidance behavior it required. The lower number of learners viewing other people's codes in Problem 3 suggests that as the difficulty level of the problem increases, the percentage of learners who view others' codes tends to decrease. Additionally, trial and error may lead to an increase in the percentage of learners who choose to view other people's codes.

6.4 Evaluation of the Group that Did Not Use the Code Viewing Control

Based on the results of the previous section, we analyze the changes in the pre- and post-test scores of the quality indicators by dividing the group into two portions, those who viewed others' codes (viewing group) and those who did not (non-viewing group).

Table 2 shows the means and standard deviations for the number of harvests obtained by the participants in this evaluation, as well as the results and effect sizes of the t-tests. We began by analyzing the results for the non-viewing group. A t-test (Student's t-test) at the 0.5% significance level shows a significant increase in the score for (α-1) the number of harvests, though some items were not significant. To compare the pre- and post-test scores within each group, a paired t-test was conducted with a significance level of 0.5% ($p < 0.005$). Additionally, the effect size (Cohen's d) was calculated to assess the practical significance of the observed differences.

For α-1 participants wrote codes that earned higher scores for the number of harvested plants. This suggests that learners were specifically trying to better the scores of their codes for the axis of the quality indicator shown by the system. This suggests that the participants were trying to improve their own codes to eventually harvest a larger crop.

6.5 Evaluation of Groups That Used the Code Viewing Function

The results for the viewing group are shown in Table 2. They show that this group significantly improved their scores on all questions for (β-1) the number of harvests, and we can see from (β-2) the effect size that the browsing group outperformed the non-viewing group in everything except for the number of harvests for Problem 1. Additionally, as mentioned above, both the browsing and non-viewing groups significantly improved their scores in terms of the number of harvests, which tells us that this system successfully encourages learners to write better code through trial and error, suggesting that having the ability to browse by ranking is not necessary for improving learner ability.

Regarding (H1), our results suggest that applying quality indicators and a ranking system to learning activities using the code visualization system is useful in teaching learners to write better code, thereby supporting this hypothesis. That code could be improved when the code browsing control function was used shows that (H2) is also supported. However, there was a significant difference in the score of (H3) due to differences in some effect sizes among the viewing groups, but no significant difference was observed among the non-viewing groups, probably due to the fact that a large difference had already occurred among the non-viewing groups. Therefore, we consider this hypothesis to be only partially supported.

Table 2. *Comparison of Pre-test and Post-test Scores*

		Harvest Points		
Problem	**Group**	**Pre-test (SD)**	**Post-test (SD)**	**p-value/Effect Size (Approx.)**
1	**Viewing**	189.4 (62.7)	899.4 (452.8)	<0.005/1.8 (Large)
	Non-viewing	62.7 (100.4)	548.0 (337.7)	<0.005/2.0 (Large)

2	**Viewing**	142.4 (16.0)	576.5 (286.7)	<0.005/1.7 (Large)
	Non-viewing	16.0 (43.3)	286.7 (321.2)	<0.005/1.2 (Large)
3	**Viewing**	149.2 (89.6)	658.8 (356.0)	<0.005/1.7 (Large)
	Non-viewing	89.6 (195.4)	356.0 (348.7)	<0.005/0.9 (Large)

6.6 Questionnaire Results

Due to space constraints, we summarize the key findings from the questionnaire. Learners rated the system's features on a six-point Likert scale (1 = "strongly disagree" to 6 = "strongly agree"). The statement "the code sharing function leads to motivated learning" received an average score of 5.1, while "the point system for programs leads to motivated learning" scored 5.0. These results suggest that the quality indicators, ranking system, and code sharing function were effective in enhancing learner motivation and engagement.

7. Conclusion

We proposed a system that motivates learners to write better code and supports the process of trial-and-error when learning programming and verified its effectiveness. Our code visualization system utilizes quality indicators, a ranking system, and code viewing control based on those rankings. Its effectiveness was evaluated in an actual lecture for second-year university students.

The experimental results suggest that having quality indicators and a ranking system is effective for encouraging learners to engage in trial and error and in helping them write better code. The quality indicators used in this study were limited to dynamic evaluation. However, the readability of code is also important, so one of our goals for future work is to propose a static quality indicator.

Acknowledgements

This study was supported by JSPS KAKENHI Grant Numbers JP22K12322 and JP21H03565.

References

Campbell, W., & Bolker, E. (2002). Teaching programming by immersion, reading and writing. *32nd Annual Frontiers in Education*, *1*, T4G23-T4G28. https://doi.org/10.1109/FIE.2002.1158015

Chen, X., Mitrovic, A., & Mathews, M. (2020). Learning From Worked Examples, Erroneous Examples, and Problem Solving: Toward Adaptive Selection of Learning Activities. IEEE Transactions on Learning Technologies, 13(1), 135–149. https://doi.org/10.1109/TLT.2019.2896080

Jiau, H. C., Chen, J. C., & Ssu, K.-F. (2009). Enhancing Self-Motivation in Learning Programming Using Game-Based Simulation and Metrics. *IEEE Transactions on Education*, 52(4), 555–562. https://doi.org/10.1109/TE.2008.2010983

Keuning, H., Heeren, B., & Jeuring, J. (2021). A Tutoring System to Learn Code Refactoring. *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 562–568. https://doi.org/10.1145/3408877.3432526

Kotsopoulos, D., Floyd, L., Khan, S., Namukasa, I. K., Somanath, S., Weber, J., & Yiu, C. (2017). A Pedagogical Framework for Computational Thinking. *Digital Experiences in Mathematics Education*, 3(2), 154–171. https://doi.org/10.1007/s40751-017-0031-2

Paiva, J. C., Leal, J. P., & Queirós, R. (2020). Game-Based Coding Challenges to Foster Programming Practice. *In First International Computer Programming Education Conference (ICPEC 2020)*. https://doi.org/10.4230/OASICS.ICPEC.2020.18

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. *Basic Books*.

Sorva, J., Karavirta, V., & Malmi, L. (2013). A Review of Generic Program Visualization Systems for Introductory Programming Education. *ACM Transactions on Computing Education*, 13(4), 1–64. https://doi.org/10.1145/2490822

Spinellis, D. (2003). Reading, Writing, and Code: The key to writing readable code is developing good coding style. *Queue*, 1(7), 84–89. https://doi.org/10.1145/957717.957782
Sweller, J. (2020). Cognitive load theory and educational technology. *Educational Technology Research and Development*, 68(1), 1–16. https://doi.org/10.1007/s11423-019-09701-3
Zhi, R., Price, T. W., Marwan, S., Milliken, A., Barnes, T., & Chi, M. (2019). Exploring the Impact of Worked Examples in a Novice Programming Environment. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 98–104. https://doi.org/10.1145/3287324.3287385