The European Proceedings of
Social and Behavioural Sciences
EpSBS

www.europeanproceedings.com e-ISSN: 2357-1330

DOI: 10.15405/epsbs.2021.02.02.126

NININS 2020

International Scientific Forum «National Interest, National Identity and National Security»

DEVELOPING AN UNDERSTANDING OF VARIABLES AND EXPRESSIONS IN INTRODUCTORY PROGRAMMING COURSES

Oleg A. Sychev (a)*, Dmitrii A. Sasov (b), Pavel A. Chechetkin (c)
*Corresponding author

(a) Volgograd State Technical University, 28, Lenin Ave, Volgograd, Russia, oasychev@gmail.com
(b) Volgograd State Technical University, 28, Lenin Ave, Volgograd, Russia, s-dima29@yandex.ru
(c) Volgograd State Technical University, 28, Lenin Ave, Volgograd, Russia, asepush@gmail.com

Abstract

Introductory programming courses often pose problems for the instructors because of the significant number of new concepts that students should learn to develop programming skills. This study is aimed at researching the efficiency of quizzing as a method to improve students' understanding of Variables topics, including variable lifetime, scope, name shadowing, name resolution, and also the variables that constitute input and output of a statement or an expression. Summative and formative quizzes with nine types of questions were created. Eighty-seven first-year computer science students attempted the summative quiz, then had two weeks to do the formative quiz as homework, then passed the summative quiz again. The results show that the students achieved statistically significant gains for the summative quiz and the students who made more than four attempts of the formative quiz gained twice as much as those who made a few attempts. Item analysis allowed identifying areas where high-performing and low-performing students made their gains. About 78 % of students stated that they want to use quizzing to learn other topics. Such quizzes, however, require significant time to create and provide explanatory feedback for students. Developing a system for automatic question generation, grading, and explanation generation is feasible. Also, students should be motivated to perform formative quizzes more.

Keywords: E-learning, introductory programming learning, quizzing, variable

https://doi.org/10.15405/epsbs.2021.02.02.126
Corresponding Author: Oleg A. Sychev
Selection and peer-review under responsibility of the Organizing Committee of the conference
eISSN: 2357-1330

1. Introduction

The concept of variable is one of the key concepts in programming basics courses. While variable seems an easy and intuitive concept, it is relatively hard to grasp fully, especially in for strongly-typed programming languages like C++, C#, and Java, including understanding lifetime and scope of variables, and the name resolution laws. The relative ease of getting acquainted with creating and using variables often leaves them understudied in the beginner's courses.

Unfortunately, introductory students often have difficulties grasping the concept of the variable (Sorva, 2008; Swidan et al., 2018). Often, their understanding of variables is limited to a few properties without learning the full scope of this concept. This limited understanding of primitive variables complicates studying more complex object variables when learning object-oriented programming and may lead to a lot of syntax and semantic errors. Sometimes, beginner programming students who are used for declaring iteration variables in loop headers unwittingly create from 4 to 7 different variables with the same name in one function, which produces bugs that are hard to fix. Developing stable programming skills requires a deeper understanding of basic concepts and learning to make responsible decisions based on them.

Several prior studies explored different methods of teaching reading and understanding program code. One popular method is the visualization of program execution which increases students' motivation and learning gains (Ishizue et al., 2018; Karavirta et al., 2015; Nelson et al., 2017). This method often allows students seeing variables creation, access, and destruction, adding to their understanding of variable as a concept. There are many programming code visualization tools. PLTutor visualizes programs on Javascript programming language with a set of examples, including visualization of such properties of variables like call frames (Nelson et al., 2017). It also may ask questions to the students about variable values after executing the next string; the visualization shows the correct answer after the question. PLTutor showed improved learning gains compared to a programming tutorial.

ViLLE is a multi-purpose exercise framework with special exercise type for program reading and visualization using JSAV open-source library (Ishizue et al., 2018) which showed increased learning gains and engagement. It shows call stack, local variables, and all variables, shared memory, and current instructions. Originally, ViLLE was designed for Python programming language assessments, but now supports more languages, including pseudocode. However, as ViLLE tries to execute Python equivalent of the code, its abilities to show the nuances of variables in other languages are limited.

Jeliot visualizes programs in Java programming language. It shows calling tree and local variables and can visualize the evaluation of expressions in details (Wang et al., 2012). A special panel in Jeliot allows watching array and object values which in Java are always stored by reference. PVC is a program execution visualizer for C programming language which has one of the most complex implementations of variables. PVC allows visualizing complex situations with the same identifier used for different variables in block scopes and name resolution algorithm. It is using PVC allowed increasing students' understanding of variables and pointers in C programming language (Ishizue et al., 2018). However, unlike PLTutor and ViLLE, Jeliot and PVC cannot question students during visualizations exercises.

https://doi.org/10.15405/epsbs.2021.02.02.126
Corresponding Author: Oleg A. Sychev
Selection and peer-review under responsibility of the Organizing Committee of the conference
eISSN: 2357-1330

Another approach to increase students' engagement and active learning is gamification. It can take the form of using a regular game that is influenced by the teacher to affect students' involvement and behaviour in class (Papadakis & Kalogiannakis, 2018). The results showed that using ClassCraft game affected engagement positively but did not affect general performance. However, gamification of the learning process does not always have a positive effect on learning. In the research of Hanus and Fox (2015), the test group using competitive context, badges, and leaderboards had lower engagement and motivation than the control group.

The serious game approach involves using games created specifically with learning goals in mind. FunJava is a game created for learning Java programming language, covering topics Basic Elements, Structured instructions, Sub programming, Recursiveness, Arrays, Files (Montes-Leon et al., 2019). FunJava consists of explaining blocks and contest blocks where questions against other students test students' understanding. Unlike ClassCraft, FunJava uses contest instead of collaboration. Sometimes, a game can have a smaller focus, for example, in the research of Adamo-Villani et al. (2013), a game was developed to help the students learn operator precedence rules.

Another popular method of teaching introductory programming courses is creating exercise for reading and analyzing code before writing it, which closes the important gap between Application and Synthesis levels in Bloom's taxonomy of learning objectives. Such exercises may include analyzing teacher-created of automatically generated code. The simplest example is reading expression code without control structures and calculating the outcome. As the research of Matsumoto et al. (2016) shows, this helps to get used for programming and developing programming skills. Another group developed Unlimited Trace Tutor, a tutoring system for automatic generation of tracing problems and showing intermediate steps of tracing (Qi & Fossati, 2020). It supports control structures and mostly generates new code by modifying parsing trees of existing code and merging them together. The performance of the volunteer students participating in the pilot experiment is improved by about 18 %. However, these generators are mostly aimed at teaching expression evaluation and control-structures execution, while the concepts of variables are explored only lightly.

Integrated development environments (IDE) may be tools for studying programming languages themselves. However, most modern development environments are too complex for the novice programmers and students of introductory programming courses. To solve this problem, in the research of Whittall et al. (2017) CodeMage, an IDE created for the learning process, was developed. Apart from supporting basic IDE function, CodeMage can detect semantic errors (given patterns for them); it contains a code generator for simplifying writing program code, a visual debugger for showing program execution, hinting system, and Narrator that helps the users learn different development tools and fix errors. CodeMage also contains tools for managing databases and demonstrating developed code to other developers. Students, working with CodeMage, said that all the advanced features of the environment were useful for code development. jGRASP is another IDE, useful for learning purposes. Its general interface is simplified, but it has enhanced debugging capabilities, including object viewer, visualizing arrays, lists, and object fields during program execution. Educational development environments may be a major tool in introductory programming courses. However, they have one major disadvantage – most of

https://doi.org/10.15405/epsbs.2021.02.02.126
Corresponding Author: Oleg A. Sychev
Selection and peer-review under responsibility of the Organizing Committee of the conference
eISSN: 2357-1330

their advanced functions are used only when a student asks the environment to use it, so some students do not use the features that would have helped them.

Another form of supporting students developing basic programming skills is by creating formative quiz assessments that guide their experience in learning basic programming concepts. In the research of Mohamed Shuhidan et al. (2011), the authors describe an experiment with a guided learning tool where volunteer students during preliminary programming course were asked seven multiple-choice questions (aimed at Knowledge and Comprehension levels of Bloom's taxonomy) and eight debugging questions (aimed at Application and Analysis levels of Bloom's taxonomy). Answering a survey, the majority of the students said that the questions were easy and helpful; they wanted more similar exercises for the other topics in the course. However, approximately half of the participants answered that these questions were simply extra work for them. Also, the volunteer participation rate was very low – 11 students out of 170 taking the course.

The formative homework quizzes may be enhanced by using questions that analyze answers and provide advanced feedback that can let students understand their mistakes and find a correct answer without teacher's interventions. For example, CorrectWriting question type can analyze open text answers on natural and formal languages and detect such mistakes as typos, missing tokens, extraneous tokens, and misplaced tokens (Sychev & Mamontov, 2018). This fact significantly improves the student's experience and saves the teacher's time. However, the mistakes determined by CorrectWriting question type are useful when teaching the syntax of a formal or natural language; their efficiency when studying other topics is limited.

Quizzing as an active learning method gained popularity (Nguyen and Mcdaniel, 2015): it increases students' reading compliance and stimulates remembering and retrieving information from the course, leading to better comprehension and increases the performance of students during final exam questions. However, this method has its pitfalls: quizzes haphazardly created from a wide range of textbook material produce no positive effect or even hinder performance. Instructors must take care to create their formative quizzes corresponding with the concepts they teach to achieve performance gain. However, creating questions aimed at applying the knowledge in different contexts is beneficial.

An important part of using quizzing as an active learning technique is that students feel motivated to perform formative quizzes during their homework, reducing instructor's workload and saving class hours for the other learning activities. However, this leaves the number of attempts of the provided formative quiz to the student's discretion, so their behaviour related to homework formative quizzes requires studying as well as quizzing efficiency.

2. Problem Statement

Current advances in automatic question generation combined with artificial intelligence methods for providing feedback allow creating automatic systems to teach topics in introductory programming courses by asking questions, grading them, and providing explanations of mistakes. However, the efficiency of using quizzes combined with automatic explanations as a learning tool and the topics where this method is useful must be determined on an objective basis. This study is aimed at determining the need and efficiency of using quizzing in learning variables and expressions in an introductory programming course for computer science students.

3. Research Questions

Specifically, we address the following research questions:

- How free using formative quizzes with feedback during homework affects the summative quiz outcome?
- Is quizzing a viable way to increase understanding of the topic Variables for first-year computer science students?
- How quizzing affects high-performing and low-performing students?

4. Purpose of the Study

The purpose of this study is to evaluate quizzing as a method of in-depth studying Variables in an introductory programming course for first-year computer science students.

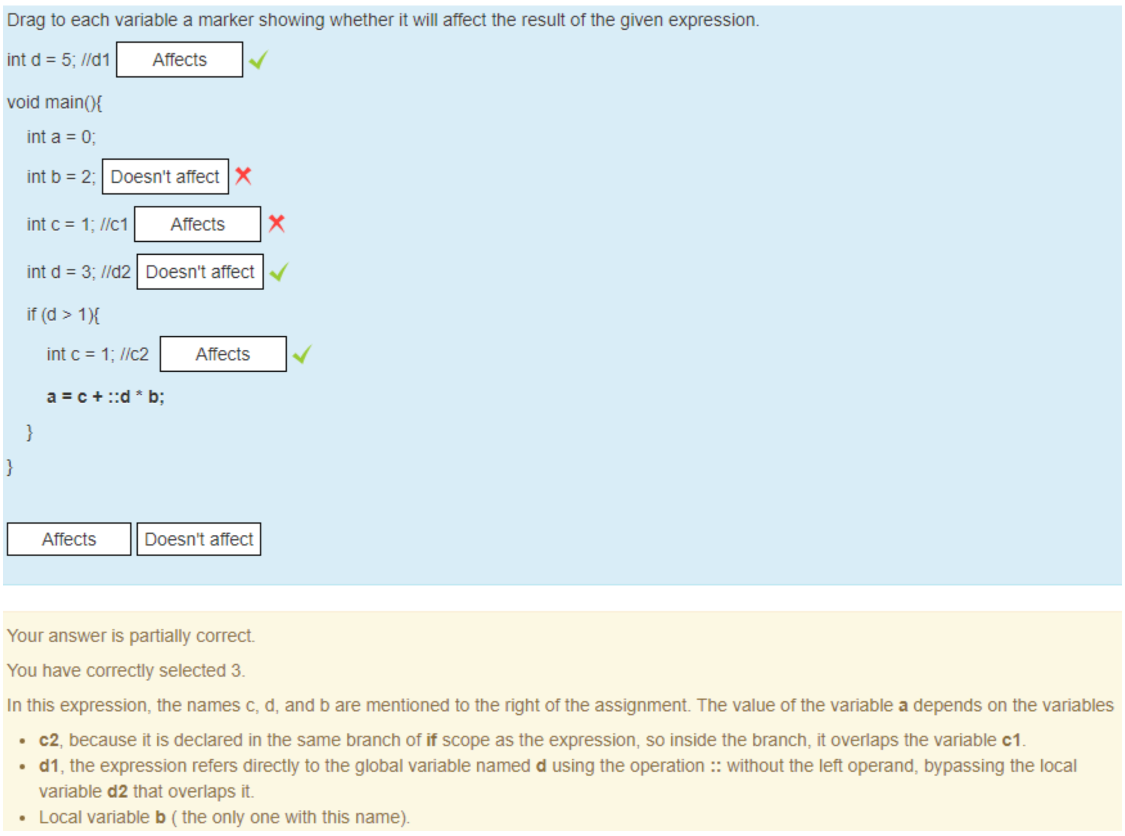
5. Research Methods

For this research, the authors created online quizzes on variables and expressions in C language regarding variables, their identifiers, and use in expressions using closed-ended multiple choice and drag-and-drop questions. The nine types of questions, developed for these tests, are provided in table 1. The original question texts were in Russian. The table contains English translations. All questions contained detailed feedback, explaining the correct answers and typical mistakes, that was shown after question completion. For multiple-choice questions, each wrong answer had feedback explaining why it is wrong. The questions used variables in global (file), function, and block scopes; some questions included variable shadowing and the scope resolution operator. Some of this information was given to the students during lectures, but more complex problems like name resolution, name shadowing, and using scope operators to access shadowed global-scope variables weren't explained before to leave the students to explore the topic on their own, using the quizzes as learning tools. The questions were divided into a training set that was used for creating formative quizzes for homework, and a holdout set that was used for classroom measure of students' performance. The training set consisted of 5–7 questions of each type. In comparison, the holdout set consisted of 15–20 questions of each type to reduce their repeating during the simultaneous classroom testing of the entire students' group. Figure 1 shows an example of a question and explanation (translated to English). The formative quiz contained one randomly selected question of each type while the summative quiz contained two randomly selected questions of each type.

https://doi.org/10.15405/epsbs.2021.02.02.126
Corresponding Author: Oleg A. Sychev
Selection and peer-review under responsibility of the Organizing Committee of the conference
eISSN: 2357-1330

Table 1. Quiz questions types

#	Topic	Question text	Question type
1	Expression input (identifiers)	Mark names which constitute the input of the given expression (i.e., their values affect its result)	Multiple choice
2	Expression output (identifiers)	Mark names which constitute the output of the given expression (i.e., their values may change after its evaluation)	Multiple choice
3	Statement output (identifiers)	Mark names whose values may change after the statement (branch condition, loop header, etc) is executed	Multiple choice
4	Name resolution	Mark the variables that are referenced by names a and b in the given expressions & Drag and drop into text	Drag and drop into text
5	Expression input (variables)	Drag to each variable a marker showing whether it will affect the result of the given expression	Drag and drop into text
6	Expression output (variables)	Drag to each variable a marker showing whether it will change after the evaluation of the given expression	Drag and drop into text
7	Variable Lifetime	In a program trace, mark the actions where variable a exists	Multiple choice
8	Statement output (variables)	Drag to each variable a marker showing whether it will change after the given statement (branch condition, loop header) is executed	Drag and drop into text
9	Variable scope	Mark the program lines where the variable is visible	Multiple choice


Drag to each variable a marker showing whether it will affect the result of the given expression.

```
int d = 5; //d1 [Affects] ✓
void main(){
    int a = 0;
    int b = 2; [Doesn't affect] ✗
    int c = 1; //c1 [Affects] ✗
    int d = 3; //d2 [Doesn't affect] ✓
    if (d > 1){
        int c = 1; //c2 [Affects] ✓
        a = c + ::d * b;
    }
}
```

[Affects] [Doesn't affect]

Your answer is partially correct.

You have correctly selected 3.

In this expression, the names c, d, and b are mentioned to the right of the assignment. The value of the variable **a** depends on the variables

- **c2**, because it is declared in the same branch of **if** scope as the expression, so inside the branch, it overlaps the variable **c1**.
- **d1**, the expression refers directly to the global variable named **d** using the operation **::** without the left operand, bypassing the local variable **d2** that overlaps it.
- Local variable **b** (the only one with this name).

Figure 1. Example of drag-and-drop-into-text question and its feedback

https://doi.org/10.15405/epsbs.2021.02.02.126
Corresponding Author: Oleg A. Sychev
Selection and peer-review under responsibility of the Organizing Committee of the conference
eISSN: 2357-1330

The quizzes were used by 87 students in fields Informatics and Computing and Software Engineering. These first-year students completed the introductory Informatics course, including programming assessments using a pedagogic programming language and started to study C programming language in Programming Basics course. They were given a basic introduction to variables in strongly-typed programming languages without direct the concepts of a variable lifetime, scope, block scope, shadowing, and name resolution. The students took the summative quiz the first time to measure their baseline understanding of variables and make them aware of the complexity of variables in the C programming language. After that, they were given access to the formative quiz for two weeks to prepare for the second attempt of the summative quiz. No limits on the attempts of the formative quiz were set, but the students were informed that the results of their second attempt at the summative quiz would affect their grades. At the end of the two-week term, they retook the summative quiz. Finishing the quizzes, the students filled a small questionnaire about their experience.

The authors measured students' performance at first (without training) and second (with training) attempts of the summative quiz. For evaluating the Discrimination Index, the students' grades for the previously completed Informatics course were used for distinguishing high-performing students from low-performing students. That allowed avoiding the test bias, caused by narrow concentration on variables and expressions, and provide more objective measure for students general performance in programming.

6. Findings

The first (baseline) attempt of the summative quiz had average score $Xc_1 = 5.16 \pm 0.5$ ($p = 0.001$) with standard deviation $SD_1 = 1.37$; for the second (after using the formative quiz) attempt of the summative quiz the average score was $Xc_2 = 6.87 \pm 0.5$with standard derivation $SD_2 = 1.37$. A paired t-test showed that this improvement is statistically significant ($t = 8.96$), which shows that using formative quizzes without teacher's intervention increases students' scores and comprehension of the concepts of variables.

The students used the formative quizzes on their own. They split into three groups: actively training (G_1, more than four formative-quiz attempts), low training (G_2, less than four formative-quiz attempts), no training (G_3, no formative-quiz attempts). Table 2 shows that while the average initial score for the actively-training group is lower than for other groups, their gains are significantly better than for the two other groups. Unpaired t-test shows that learning gains differs significantly in pairs $G_1 – G_2$ ($t = 4.61$) and $G_1 – G_3$ ($t = 6.54$), however G_2 and G_3 gains are not significantly different with $t = 1.53$. So actively training students performed the second time significantly better than others while having a similar initial score.

Table 2. Students by using the formative quiz

Number of training attempts	Average first-attempt score	Average gain
5–13	4.82	3.63±0.83
1–3	5.19	1.83±0.79
0	5.27	1.26±0.65

https://doi.org/10.15405/epsbs.2021.02.02.126
Corresponding Author: Oleg A. Sychev
Selection and peer-review under responsibility of the Organizing Committee of the conference
eISSN: 2357-1330

Table 3 shows the results of students' performance for each question type. It can be seen that students showed a statistically significant increase for most question types. However, changes in the discrimination index for the first and the second attempts indicate that different kinds of students learned better in different types of questions. For question types 2, 5, and 9 discrimination index dropped significantly, showing progress made mostly by low-performing students. However, for question types 6 and 8 discrimination index rose, showing progress made mostly by high-performing students. Finally, question types 1 and 7 show significant score gains with relatively stable discrimination index, showing consistent gains for all the students.

Filling the questionnaire, 26.5 % of students marked the option that they want to study other topics using quizzing, 52 % marked the option to combine quizzing with classroom studying of programming, and 21.5 % preferred to study programming using only traditional classroom methods.

Table 3. Item analysis for different question types

#	Topic	Average score (1st attempt)	Average score (2nd attempt)	Discrimination Index (1st attempt)	Discrimination Index (2nd attempt)
1	Expression input (identifiers)	0.28±0.096	0.53± 0.111	0.09	0.07
2	Expression output (identifiers)	0.62±0.125	0.81±0.093	0.33	0.15
3	Statement output (identifiers)	0.53±0.115	0.69±0.093	0.13	0.09
4	Name resolution	0.59±0.083	0.79±0.068	0.15	0.13
5	Expression input (variables)	0.73±0.065	0.86±0.050	0.13	0.02
6	Expression output (variables)	0.62±0.054	0.77±0.054	0.07	0.19
7	Variable Lifetime	0.29±0.103	0.47±0.113	0.19	0.13
8	Statement output (variables)	0.63±0.071	0.80±0.061	0.04	0.22
9	Variable scope	0.35±0.073	0.52±0.088	0.09	0.00

7. Conclusion

Our findings show that using formative quizzes gives statistically significant learning gains in an introductory programming course; for the active training students, the gains are about two times higher than for those who made less than four attempts. The important moment is that actively training students had a subtly lower average score on the first attempt, so the better success of this group cannot be caused by their fundamental knowledge or naturally high performance. The majority of the students gave positive feedback and stated that they want to continue to learn through quizzing either on itself or in combination with more traditional methods. The students commented that this technique let them learn more about variables than the classroom activities allowed. Also, performing quizzes made the active students participants in the learning process which significantly increases their engagement compared to just receiving the information.

Considering item analysis, we can see that for the questions about expression input, variable scope, and expression output regarding identifiers, low-performing students gained most. In contrast, for both questions about output regarding variables, the formative quiz was mostly useful for high-performing students. Each group had gains in different areas, but the formative quiz was useful for everyone. There is no use in restricting formative quizzes to specific questions for different groups of students, however, as their results do not affect grades. Hence, students are free to concentrate only on the questions they expect to learn more from.

The study results show that quizzing is an efficient method of learning concepts in an introductory programming course, specifically in-depth learning of the concepts regarding such basic entities as variables and identifiers. Developing a system allowing automatic question generation, grading and providing explanatory feedback is feasible as it lessens an instructor's workload on creating questions and providing feedback to the students who encounter problems. Future work will address the issues of creating such a system.

However, the number of actively training students was relatively low while their gains were significantly better. So more efforts on studying and improving students' motivation should be spent. Preliminary research shows that the number of attempts of formative quizzes is higher when passing a certain threshold of the summative quiz score was required for course completion compared to the situation where the summative quiz affected the final grade but had no required threshold.

Acknowledgments

The reported study was funded by RFBR, project number 20-07-00764.

References

Adamo-Villani, N., Haley-Hermiz, T., & Cutler, R. (2013). Using a Serious Game Approach to Teach 'Operator Precedence' to Introductory Programming Students. *17th Int. Conf. on Inform. Visualisat.* (pp. 523–526). London: Instit. of Electr. and Electr. Engin. https://doi.org/10.1109/IV.2013.70

Hanus, M. D., & Fox, J. (2015). Assessing the effects of gamification in the classroom: A longitudinal study on intrinsic motivation, social comparison, satisfaction, effort, and academic performance. *Comput. and Ed., 80*, 152–161. https://doi.org/10.1016/j.compedu.2014.08.019

Ishizue, R., Washizaki, H., Sakamoto, K., & Fukazawa, Y. (2018). PVC: Visualizing C programs on web browsers for novices. *Proc. of the 49th ACM Techn. Symp. on Computer Sci. Ed. (SIGCSE 2018)*, vol. 2018-January. (pp. 245–250). Baltimore: Associat. for Comput. Machin. https://doi.org/10.1145/3159450.3159566

Karavirta, V., Haavisto, R., Kaila, E., Laakso, M.-J., Rajala, T., & Salakoski, T. (2015). Interactive learning content for introductory computer science course using the ViLLE exercise framework. *Proc. Int. Conf. on Learn. and Teach. in Comput. and Engineer.* (pp. 245–250). Taipei: Instit. of Electr. and Electr. Engin. https://doi.org/10.1109/LaTiCE.2015.24

Matsumoto, S., Okimoto, K., Kashima, T., & Yamagishi, S. (2016). Automatic generation of C source code for novice programming education. *Lect. Notes in Computer Sci., 9731*, 65–76. https://doi.org/10.1007/978-3-319-39510-4_7

Mohamed Shuhidan, S., Hamilton, M., & D'Souza, D. (2011). Understanding novice programmer difficulties via guided learning. *Proc. of the 16th Annual Joint Conf. on Innovat. and Technol. in Computer Sci. Ed.* (pp. 213–217). Darmstadt: Associat. for Comput. Machin. https://doi.org/10.1145/1999747.1999808

Montes-Leon, H., Hijon-Neira, R., Perez-Marin, D., & Leon, S. R. M. (2019). Improving Programming Learning on High School Students through Educative Apps. *Int. Symp. on Comput. in Ed. (SIIE)* (pp. 1–6). Tomar: Instit. of Electr. and Electr. Engin. https://doi.org/10.1109/SIIE48397.2019.8970112

Nelson, G. L., Xie, B., & Ko, A. J. (2017). Comprehension first: Evaluating a novel pedagogy and tutoring system for program tracing in CS1. *Proc. of the 2017 ACM Conf. on Int. Comput. Ed. Res.* (pp. 2–11). Tacoma: Associat. for Comput. Machin. https://doi.org/10.1145/3105726.3106178

Nguyen, K., & Mcdaniel, M. A. (2015). Using Quizzing to Assist Student Learning in the Classroom: The Good, the Bad, and the Ugly. *Teach. of Psychol., 42,* 87–92. https://doi.org/10.1177/0098628314562685

Papadakis, S., & Kalogiannakis, M. (2018). Using gamification for supporting an introductory programming course. The case of classcraft in a secondary education classroom. *Lect. Notes of the Instit. for Comput. Sci., Soc.-Inform. and Telecommunicat. Engin., 229,* 366–375. https://doi.org/10.1007/978-3-319-76908-0_35

Qi, R., & Fossati, D. (2020). Unlimited trace tutor: Learning code tracing with automatically generated programs. *Proc. of the 51th ACM Techn. Symp. on Computer Sci. Ed.* (pp. 427–433). Portland: Associat. for Comput. Machin. https://doi.org/10.1145/3328778.3366939

Sorva, J. (2008). The same but different: Students' understandings of primitive and object variables. *Proc. of the 8th Int. Conf. on Comput. Ed. Res.* (pp. 5–15). Koli: Associat. for Comput. Machin. https://doi.org/10.1145/1595356.1595360

Swidan, A., Hermans, F., & Smit, M. (2018). Programming misconceptions for school students. *Proc. of the ACM Conf. on Int. Comput. Ed. Res.* (pp. 151–159). Espoo: Associat. for Comput. Machin. https://doi.org/10.1145/3230977.3230995

Sychev, O. A., & Mamontov, D. P. (2018). Automatic Error Detection and Hint Generation in the Teaching of Formal Languages Syntax Using Correctwriting Question Type for Moodle LMS. *3th Russian-Pacific Conf. on Computer Technol. and Applicat. (RPC)* (pp. 1–4). Vladivostok: Instit. of Electr. and Electr. Engin.

Wang, P., Bednarik, R., & Moreno, A. (2012). During Automatic Program Animation, Explanations after Animations Have Greater Impact than before Animations. *Proc. of the 12th Koli Calling Int. Conf. on Comput. Ed. Res.* (pp. 100–109). New York: Associat. for Comput. Machin. https://doi.org/10.1145/2401796.2401808

Whittall, S. J., Prashandi, W. A. C., Himasha, G. L. S., De Silva, D. I., & Suriyawansa, T. K. (2017). CodeMage: Educational programming environment for beginners. *9th Int. Conf. on Knowledge and Smart Technol. (KST)* (pp. 311–316). Chonburi: Instit. of Electr. and Electr. Engin. https://doi.org/10.1109/KST.2017.7886101