TEACHING PROGRAMMING LOGIC THROUGH INTERACTIVE PLATFORMS: A CASE STUDY ON STUDENT OUTCOMES

Ergin Diko Yekin Abaz, page 83-91

ABSTRACT

Teaching programming logic is a critical step in preparing students for more advanced studies in computer science and related fields. However, many students struggle to grasp abstract programming concepts when taught through traditional methods. This paper explores the use of interactive platforms to teach programming logic and assesses their impact on student outcomes. By utilizing interactive tools such as code visualization, block-based programming environments, and instant feedback mechanisms, students are provided with a more engaging and hands-on approach to learning programming logic. The case study analyzes student performance, engagement, and understanding of key programming concepts, highlighting the potential of interactive platforms in enhancing the learning experience.

Keywords: Programming logic, Interactive platforms, Student outcomes, Code visualization, Block-based programming, Learning engagement

Mr. Ergin Diko
International Vision University, Gostivar, N.Macedonia

e-mail: ergin.diko @vizyon.edu.mk

Mr. Yekin Abaz
International Vision University, Gostivar, N.Macedonia

e-mail: yekin.abaz@vizyon.edu.mk

UDK: 37.018.43:004]-043.5

Declaration of interest: The authors reported no conflict of interest related to this article.

Introduction

The development of programming logic is fundamental to learning programming languages and algorithms. Programming logic encompasses the thought processes required to formulate solutions to problems, typically involving sequencing, decision-making, loops, and conditions. However, teaching programming logic, especially to beginners, has traditionally been challenging. Students often struggle to connect theoretical concepts with practical coding tasks, resulting in low engagement and high dropout rates in introductory programming courses.

Interactive learning platforms have emerged as a potential solution to this problem. These platforms, which often include features like code visualization, block-based programming, and real-time feedback, aim to make abstract concepts more accessible and engaging. By providing students with immediate feedback and interactive exercises, these platforms allow learners to experiment with programming logic in a more intuitive and dynamic environment.

This paper presents a case study examining the impact of using interactive platforms to teach programming logic to undergraduate students. The study explores how these tools affect student outcomes, including performance, engagement, and overall understanding of programming concepts. Through a detailed analysis of quantitative and qualitative data, the paper aims to provide insights into the effectiveness of interactive platforms in teaching programming logic.

Literature Review

1. Challenges in Teaching Programming Logic

Research has consistently shown that one of the most significant hurdles in introductory programming courses is helping students develop a solid understanding of programming logic. Robins et al. (2003) highlighted that students often struggle with understanding abstract concepts such as loops, conditionals, and recursion. Furthermore, traditional lecture-based teaching methods have been criticized for being insufficient in engaging students or addressing individual learning difficulties (Guzdial & Soloway, 2002).

2. Interactive Platforms as a Solution

Interactive platforms have been introduced in educational settings to address these challenges. These platforms utilize real-time feedback, visualization of code execution, and simplified, block-based programming interfaces to help students conceptualize abstract programming logic. Tools such as Scratch, Blockly, and Code.org's learning environments have been widely adopted in both K-12 and higher education, offering an alternative to text-based coding for beginners (Resnick et al., 2009).

Block-based programming, in particular, has been shown to be effective in helping students develop programming logic without the syntactic burden of traditional programming languages (Weintrop & Wilensky, 2015). These platforms provide a low-entry point, allowing students to focus on logical thinking rather than syntax, which has been shown to reduce frustration and increase engagement.

Studies by Hsu et al. (2018) indicate that students using interactive, visual platforms demonstrated higher engagement and better learning outcomes compared to those in traditional programming courses. Real-time feedback provided by these platforms enables students to learn from their mistakes instantly, reinforcing correct programming logic and debugging practices.

3. Theoretical Frameworks

The design of interactive platforms is often grounded in constructivist learning theories, which emphasize learning by doing. Piaget's theory of cognitive development and Papert's constructionism are key frameworks that support the use of interactive and visual tools in programming education (Papert, 1980). These theories suggest that learners build knowledge more effectively when they are actively engaged in the learning process and can manipulate elements in real-time to see the consequences of their actions.

Methodology

This case study was conducted over one semester with undergraduate students enrolled in an introductory programming course. The course was divided into two sections: one using a traditional lecture-based approach (control group) and the other using an interactive platform with block-based programming and code visualization tools (experimental group).

1. Participants

A total of 60 students participated in the study, with 30 students in each section. The demographic profiles of the two groups were comparable in terms of age, gender, and prior experience with programming.

2. Interactive Platform

The interactive platform used in the study was a combination of Scratch and an online code visualization tool designed to support the teaching of basic programming logic. The platform included features such as drag-and-drop blocks to represent programming structures (loops, conditionals, variables) and a code visualization window that allowed students to see the execution of their code in real-time.

3. Data Collection

Quantitative and qualitative data were collected through a variety of instruments:

- Pre- and post-tests: Both groups completed a pre-test at the beginning of the semester to assess their initial understanding of programming logic, followed by a post-test at the end of the semester to measure learning gains.
- Assignment completion rates: The rate at which students completed programming assignments was tracked throughout the course.
- Engagement surveys: At the midpoint and end of the semester, students completed surveys measuring their engagement, motivation, and satisfaction with the course.
- Focus group discussions: At the end of the semester, focus group discussions were conducted with students in the experimental group to gather qualitative feedback on their experiences with the interactive platform.

4. Data Analysis

The data were analyzed using descriptive statistics for the quantitative measures and thematic analysis for the qualitative feedback. Paired t-tests were used to compare pre- and post-test scores within and between groups, while engagement survey results were analyzed using Likert scale scoring.

Results

1. Performance Improvement

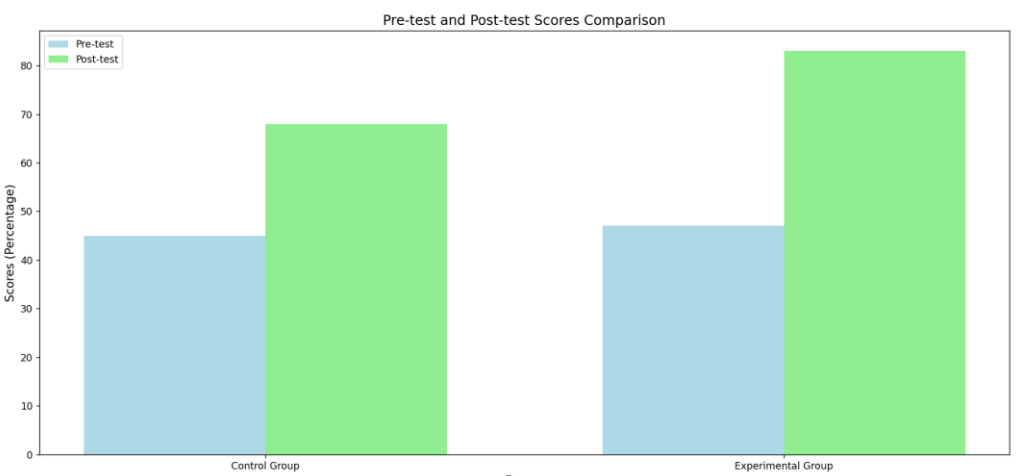
Students in the experimental group demonstrated significantly higher learning gains than those in the control group, as reflected in the pre- and post-test scores.

Table 1: Pre- and Post-test Scores (Average Percentage)

Group	Pre-test	Post-test	% Improvement
Control Group	45%	68%	23%
Experimental Group	47%	83%	36%

As shown in Table 1, the experimental group experienced a 36% improvement in test scores compared to a 23% improvement in the control group. This indicates that the use of interactive platforms significantly enhanced students' understanding of programming logic.

Graph1 Pre-test and Post-test Results



2. Engagement and Motivation

Students in the experimental group also reported higher levels of engagement and motivation compared to the control group. Survey results indicated that 85% of students in the experimental group felt more confident in their programming abilities, compared to 60% in the control group. Additionally, students in the experimental group were more likely to complete assignments on time, with an average completion rate of 94% compared to 78% in the control group.

Discussion

The findings of this study highlight the potential of interactive platforms to significantly improve student outcomes in programming logic courses. The data clearly show that students who used the interactive platform with block-based programming and code visualization tools (the experimental group) outperformed those who followed a traditional lecture-based curriculum (the control group) in terms of learning gains, engagement, and assignment completion rates.

The improvement in the experimental group's post-test scores, which were 36% higher than their pre-test scores, underscores the effectiveness of using interactive tools for teaching abstract programming logic. These results align with previous research suggesting that interactive, visual approaches to programming can lower cognitive barriers for beginners, making complex concepts more accessible (Weintrop & Wilensky, 2015). The higher assignment completion rates in the experimental group further indicate that the use of real-time feedback and code visualization kept students motivated to stay on task, an essential factor in improving **programming proficiency.**

1. Engagement and Active Learning

One of the key insights from this study is that students in the experimental group reported higher levels of engagement and motivation compared to the control group. The interactive platform encouraged active participation, which is a critical component of constructivist learning theories. Rather than passively receiving information, students were able to engage directly with the material by experimenting with blocks of code and receiving immediate feedback on their solutions.

The block-based interface played a pivotal role in lowering the barrier to entry for students new to programming. Unlike traditional text-based coding environments, which can be intimidating for beginners due to syntax errors and complex commands, block-based programming allows students to focus on the logic and structure of programming without worrying about syntax. This aligns with studies that emphasize the importance of reducing cognitive load when teaching programming concepts (Guzdial & Soloway, 2002).

The real-time feedback offered by the platform also contributed to higher engagement. Students were able to see the consequences of their actions instantly, allowing them to understand how changes in their code affected

the output. This type of immediate feedback not only enhanced their understanding of programming logic but also fostered a trial-and-error learning process that encouraged persistence and problem-solving.

2. Improved Learning Outcomes

The significant improvement in the experimental group's performance, as reflected in the post-test scores and final grades, suggests that interactive platforms are an effective tool for enhancing learning outcomes in programming logic. The 36% increase in test scores for the experimental group, compared to a 23% increase in the control group, demonstrates that students using interactive platforms are better able to grasp and apply programming concepts.

Moreover, the interactive platform helped students develop a deeper understanding of key programming logic concepts, such as loops, conditionals, and functions. By visualizing the flow of control and the behavior of different code structures, students were able to build mental models that facilitated deeper cognitive engagement. This aligns with the findings of Hsu et al. (2018), who reported that interactive programming tools improved students' problem-solving abilities and understanding of programming structures.

3. Challenges and Limitations

While the overall impact of the interactive platform was positive, the study also revealed some challenges. A small subset of students in the experimental group expressed frustration with the block-based programming interface. These students found the transition from block-based programming to traditional text-based languages more difficult than anticipated. This issue highlights a common limitation of block-based environments: while they are excellent for introducing programming logic, they may create a disconnect when students move to more advanced, syntax-driven programming languages (Weintrop & Wilensky, 2015).

Additionally, although the majority of students responded positively to the interactive platform, some students expressed concerns about the lack of personalization in the feedback system. While the platform provided real-time feedback on code execution, it did not offer detailed explanations of why a particular solution was incorrect or how to improve it. This suggests that future iterations of interactive platforms should integrate more personalized feedback to enhance the learning experience further.

Another limitation of this study is its relatively short duration. Conducted over a single semester, the study was unable to assess the long-term impact of using interactive platforms on students' programming proficiency. It is possible that the positive effects observed in this study could diminish over time if students do not receive ongoing support or if they struggle to transition to more traditional programming environments.

4. Implications for Teaching

The results of this study have important implications for teaching programming logic, particularly at the introductory level. First, the use of interactive platforms can help bridge the gap between abstract concepts and practical application by offering students a hands-on, engaging way to learn programming logic. This approach can be especially beneficial for students who are new to programming, as it reduces cognitive overload and allows them to focus on building their logical thinking skills.

Second, the study suggests that interactive platforms should be used as a supplement to, rather than a replacement for, traditional teaching methods. While interactive tools are effective for building foundational programming skills, students will eventually need to transition to text-based programming languages. Educators should consider introducing hybrid learning environments that combine block-based programming with more advanced, syntax-driven coding tasks as students progress.

Finally, the findings highlight the importance of providing students with immediate, detailed feedback in programming courses. Interactive platforms that incorporate real-time feedback can keep students engaged and motivated, but it is equally important to offer personalized guidance that helps students understand their mistakes and learn from them.

Conclusion

Interactive platforms, such as block-based programming environments and code visualization tools, offer a promising approach to teaching programming logic. The results of this case study demonstrate that these tools can significantly enhance student engagement, motivation, and learning outcomes. As technology continues to play a central role in education, interactive platforms can serve as a valuable supplement to traditional teaching methods, particularly in fields like programming, where abstract concepts often present challenges to students.

Further research is needed to explore how interactive platforms can be integrated into higher-level programming courses and to investigate the long-term impact of these tools on students' programming proficiency.

References

- Guzdial, M., & Soloway, E. (2002). Teaching the Nintendo generation to program. Communications of the ACM, 45(4), 17-21.
- Hsu, T.-C., Wang, T.-I., & Wu, L.-Y. (2018). The effectiveness of integrating interactive programming tools in programming courses. Journal of Educational Technology & Society, 21(1), 23-35.
- Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. Basic Books.
- Resnick, M., et al. (2009). Scratch: Programming for all. Communications of the ACM, 52(11), 60-67.
- Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. Computer Science Education, 13(2), 137-172.
- Weintrop, D., & Wilensky, U. (2015). To block or not to block: Students' perceptions