Majumdar, R. et al. (Eds.) (2025). Proceedings of the 1st International Conference on Learning Evidence and Analytics. Asia-Pacific Society for Computers in Education

Examining Metacognitive Difficulties in Learning Programming: Analysis of Student Behavior and Strategy

Huiyong LI[a], Boxuan MA[b*] & Chengjiu YIN[a]
[a]*Research Institute for Information Technology, Kyushu University, Japan*
[b]*Faculty of Arts and Science, Kyushu University, Japan*
*boxuan@artsci.kyushu-u.ac.jp

Abstract: Novice programmers often struggle to articulate their problem-solving steps, underscoring the need for metacognitive awareness and self-regulated learning strategies. However, most automated assessment tools in university programming courses do not effectively support these essential skills. To address this gap and guide the design of more supportive assessment tools, this study aims to investigate the metacognitive challenges novice learners face when learning Python programming, focusing on both their behavioral patterns and metacognitive strategies. The significant differences in students' program development behaviors and metacognitive learning strategy use between novice and advanced programmers were identified. Our findings highlight the need for better-designed automated assessment tools in the future.

Keywords: Introductory programming, self-regulated learning, metacognition, learning analytics, automated assessment tools

1. Introduction

Programming is a complex task that demands not only syntactic and algorithmic knowledge but also the ability to plan, monitor, and reflect one's thinking—essential skills encompassed by metacognitive and self-regulated learning (SRL) strategies, which are known to enhance learners' lifelong learning ability (Li & Ma, 2025). Nonetheless, many programming beginners struggle to apply these strategies effectively (Bergin et al., 2005). This leads to relying on trial-and-error methods that bypass deeper reflection on the problem, resulting in confusion or misinterpretation of requirements (Loksa et al., 2016). Additionally, many university courses now utilize Automated Assessment Tools (AATs) that provide immediate error messages and pass/fail indicators. While convenient, these narrow feedback loops do little to foster metacognitive growth as students can easily become preoccupied with correcting superficial mistakes (Loksa & Ko, 2016). This gap highlights a critical need for instructional and technological support that can scaffold learners' metacognition and self-regulation.

Although recent studies have begun using LLMs to generate reflective prompts for learners (Gong et al., 2024; Li & Ma, 2025), it is difficult to build truly useful assistant tools without a solid understanding of the strategies they employ and the specific obstacles they encounter. This paper aims to examine how students approach coding tasks, highlighting the range of strategies they employ and the specific obstacles they encounter. Our findings will serve as a foundation for designing AATs that not only test the correctness of learners' code but also cultivate their metacognitive learning skills by reflectively solving problems.

2. Method

2.1 Participants

This study was conducted in a laboratory environment. A total of 22 students (14 males and 8 females, with an average age of 24) were recruited from a national university in Japan. The students were from different departments. Their programming experience ranges from zero months to 9 years. Informed consent was obtained from all participants for this study.

Table 1. *Program development behavior from novice and advanced programming learners*

Group	N	Program creation (ns)	Checking solution (ns)	Checking solution with unpassed *	Checking solution with passed (ns)
Novice	14	19.07	12.5	9.5	3
Advanced	8	15.88	8.0	4.25	3.75

2.2 Learning Tasks and Study Design

Students were invited to enroll in an introductory Python programming course in a Moodle Learning Management System (LMS). They were asked to read the lecture's introductory materials, which covered fundamental concepts in Python programming (e.g., basic functions) and solve four beginner-level coding questions using CodeRunner, an open-source Moodle plugin automated assessment tool. Students were given the flexibility to complete the task at their own pace within a 35-minute learning session. They could read the problem statement, input their code, and submit their solutions to be automatically checked against pre-defined test cases. All reading and interaction activities were logged automatically by the Moodle platform. After completing the learning session, all students participated in face-to-face semi-structured interviews, each lasting approximately 30 minutes. The interviews focused on three main areas: planning ("How do you plan to solve the programming questions?"), self-reflection ("What would you do after submitting your code solutions?"), and overall experience ("Which aspects were successful, and which areas require improvement?").

2.3 Data Analysis

Students were divided into novice (n = 14) and advanced (n = 8) groups based on their prior programming experience. Students' interactions throughout the programming tasks were collected, including program creation, checking solution with unpassed and with passed. A thematic analysis was conducted on interview responses, using a coding scheme adapted from prior SRL research in programming (Silva et al., 2024) to identify metacognitive strategies.

Three analyses between the two groups were conducted in this study: (1) behavioral differences during programming were analyzed using Wilcoxon rank-sum or t-tests after normality testing; (2) the frequency of planning and self-reflection strategies was compared; and (3) key metacognitive difficulties were extracted based on interview.

3. Results

3.1 Differences in Program Development Behavior

Table 1 presents the differences in program development behavior between the two groups. The average number of program creation actions was slightly higher for novice learners (M = 19.07) than for advanced learners (M = 15.88), but the difference was not statistically significant (W = 69.5, p = .373). Similarly, while novice learners checked their solutions more frequently (M = 12.5) than advanced learners (M = 8.0), this difference did not reach significance (t = 1.98, p = .062). However, a separate analysis revealed that novices exhibited significantly more checking behavior with unpassed (t = 2.13, p = .046), with a large effect size (Cohen's d = 0.822).

3.2 Differences in Metacognitive Learning Strategy Use

Table 2 summarizes the metacognitive learning strategies used by the two groups. Overall, novices employed fewer strategies than their advanced peers across all seven strategy types. Among novices, the most frequently used strategies were P1: understanding the problem and R3: code review (both 42.9%). In contrast, advanced learners more frequently applied P1 (87.5%), P3: program logic planning (75%), R3 (87.5%), and R4: code optimization (50%), indicating a broader and deeper use of metacognitive strategies.

Table 2. *Frequency and proportion of strategy use by novice and advanced learners*

Strategy	Count		Proportion	
	Novice (n=14)	Advanced (n=8)	Novice (n=14)	Advanced (n=8)
P1	6	7	42.9%	87.5%
P2	2	3	14.3%	37.5%
P3	2	6	14.3%	75%
R1	3	2	21.4%	25%
R2	1	0	7.1%	0%
R3	6	7	42.9%	87.5%
R4	0	4	0%	50%

* P1: Understanding the Problem; P2: Problem definition; P3: Program logic planning; R1: Achievement self-assessment; R2: Effort self-assessment; R3: Code review; R4: Code optimization.

3.3 Metacognitive Difficulty Extracted from Interview

According to interviews, novices commonly struggle with misunderstandings of problem descriptions, syntax, and semantic comprehension, as well as unfamiliarity with problem-solving in programming, unawareness of knowledge gaps, limited error interpretation skills, and poor time and resource management. In contrast, advanced learners reported issues such as misunderstanding error messages, insufficient attention to foundational knowledge (e.g., mathematical concepts), and overreliance on code assistance tools.

4. Conclusion

This study examines the metacognitive challenges that learners face when studying programming, with a focus on SRL behavioral patterns and strategies. Our findings show that novice learners employ fewer and less diverse metacognitive strategies than advanced learners, particularly in areas such as program logic planning and code optimization. Interview data further revealed that novices struggle with comprehending problems, as well as interpreting errors and managing their time. At the same time, advanced learners face issues such as overlooking fundamental knowledge. These findings suggest both the potential and the necessity of integrating SRL support into AATs, especially improved support for error interpretation, structured scaffolds for planning, and better prompts for reflection.

Acknowledgments

This work is supported by JSPS KAKENHI Grant Numbers JP24K20903 and JP25K17078.

References

Bergin, S., Reilly, R., & Traynor, D. (2005, October). Examining the role of self-regulated learning on introductory programming performance. In *Proceedings of the first international workshop on Computing education research* (pp. 81-86).

Gong, X., Li, Z., & Qiao, A. (2024). Impact of generative AI dialogic feedback on different stages of programming problem solving. *Education and Information Technologies*, 1-21.

Li, H., & Ma, B. (2025). Design of AI-Powered Tool for Self-Regulation Support in Programming Education. *arXiv preprint arXiv:2504.03068*.

Loksa, D., Ko, A. J., Jernigan, W., Oleson, A., Mendez, C. J., & Burnett, M. M. (2016). Programming, problem solving, and self-awareness: Effects of explicit guidance. In *Proceedings of the 2016 CHI conference on human factors in computing systems* (pp. 1449-1461).

Loksa, D., & Ko, A. J. (2016). The role of self-regulation in programming problem solving process and success. In *Proceedings of the 2016 ACM conference on international computing education research* (pp. 83-91).

Silva, L., Mendes, A., Gomes, A., & Fortes, G. (2024). What Learning Strategies are Used by Programming Students? A Qualitative Study Grounded on the Self-regulation of Learning Theory. *ACM Transactions on Computing Education, 24*(1), 1-26.