Shih, JL. et al. (Eds.) (2023). Proceedings of the 31st International Conference on Computers in Education. Asia-Pacific Society for Computers in Education

Measuring Self-regulated Learning Processes in Computer Science Education

Elizabeth B. CLOUDE[a*], Ryan S. BAKER[a] & Maciej PANKIEWICZ[b]
[a]Penn Center for Learning Analytics, University of Pennsylvania, USA
[b]Institute of Information Technology, Warsaw University of Life Sciences, Poland
*ecloude@upenn.edu

Abstract: Self-regulated learning (SRL) is important for computer science education. Yet, students often do not have SRL skills to benefit their learning. In this study, we examined 187 (*n*=187) students' SRL behaviors while they built programs with an automated feedback tool. Anchored in Winne and Hadwin's (1998) COPES model of SRL, our results showed that novices used more operators to debug compiler errors, while more experienced programmers used more operators to debug non-compiler errors. Finally, a random forest classifier showed that prior knowledge was the most important COPES feature predicting learning gain, followed closely by the student's perceived programming ability, use of evaluations with the automated feedback tool, and operators used to debug non-compiler errors on failed programs.

Keywords: Computer Science Education, Self-regulated Learning, Learning

1. Introduction

Computer science (CS) education requires students to develop self-regulated learning (SRL) skills (Prather et al., 2018); yet many students, particularly novices, do not have the SRL skills to benefit their performance (Arakawa et al., 2021; Loksa et al., 2016). SRL requires monitoring and adapting learning processes and strategies to purse a goal. COPES (Winne & Hadwin, 1998), a widely adopted framework, emphasizes that feedback is the central mechanism driving SRL, adaption, and learning.

Many CS educators rely on automated assessment tools since they offer automatic feedback and evaluation (Chen et al., 2020), creating a feedback loop between the student and automated tool, possibly opening doors to support SRL. Arakawa et al. (2021) examined whether students struggled to initiate SRL as they programmed with an automated feedback tool. Results showed that often students ignored the feedback from the tool altogether, perhaps due to a lack of SRL skills (Arakawa et al., 2021). Loksa and others (2016) examined relations between coding errors and SRL strategies collected using a think-aloud protocol. While students programmed with an automated feedback tool, the results showed novices used SRL infrequently and often ineffectively for debugging errors. In contrast, experienced programmers used SRL during programming and they made fewer errors. Marwan et al. (2022) and Ko et al. (2019) found similar results to Loksa et al. (2016). Novice and experienced students differed in their use of SRL strategies and coding errors with an automated feedback tool. In sum, prior programming knowledge plays a role on how SRL manifests with automated feedback tools and its relation to code quality. More research is needed to explore relations between prior knowledge, SRL processes with an automated feedback tool, and learning outcomes in introductory programming.

2. Methods

Data were collected from 245 CS undergraduates (*n*=245) during a mandatory, first-semester programming course (C#) at a large University in Poland. Due to missing data for interest variables, students (*n*=58) were removed from our analysis, resulting in a subset of

187 students (*n*=187; 39% female). The course covered 146 tasks on 8 basic CS education topics. The automated assessment tool, runcodeapp.com (Pankiewicz, 2020) allowed students to submit code on a pre-defined set of test cases. Feedback was provided as a total score, compiler errors, and detailed results for each test case. To view detailed feedback for each test case, students had to click on a selected test case.

Participants completed a 5-point Likert scale (1=low, 5=high) to report their programming ability. An 8-item, multiple-choice pre-test was also administered to measure general programming knowledge (*Mdn*=38.75). The Likert scale and pre-test measures were used to assign students to a novice or more experienced group. A Wilcoxon test found pre-test scores were significantly higher for students who reported more programming experience (*Mdn*=65.88) than novices (*Mdn*=16.25), *W*=8079, *p*<.001. As such, students were assigned to the more experienced group (*n*=87) if they reported a higher programming ability (3-5), or to the novice group if low (1 or 2; *n*=100). A 9-item, multiple-choice test evaluating knowledge of the course's programming topics was administered at the middle (*Mdn*=55.56) and end of the semester (*Mdn*=66.67). Normalized learning gain (NLG) was used to calculate a normalized score of the max possible change from mid- to post-test (*M*=.09, *SD*=.38; see Marx & Cummings, 2007). To explore SRL with an automated feedback tool and its relation to learning, we grounded our work in Winne and Hadwin's (1998) COPES model (see Table 1). Log data on code and interactions with the automated feedback tool were collected.

Table 1. COPES construct descriptions and operational definitions.

COPES	Description	Operational definitions
Conditions	Cognitive or external resources	• Self-reported programming ability. • General programming knowledge scores.
Operators	Primitive information processing mechanics	The proportion of consecutive test cases for code that scored <100: contained either a) compiler or b) non-compiler errors.
Products	Knowledge shaped by operators	Not analyzed in current study.
Evaluations	Feedback on discrepancies between products & standards	Average clicks on results of resubmitted code from the automated feedback tool.
Standards	The criteria of success	Criteria for perfect program solutions.

3. Results

Do novices and more experienced programmers differ in the evaluations and operators used to debug failed programs with compiler and non-compiler errors?

The Benjamini and Hochberg method controlled for type I errors. A Mann-Whitney test found more experienced (*Mdn*=.5) and novice programmers (*Mdn*=.63) marginally differed in how many operators they used to debug compiler errors, *W*= 3687, adj. α=.05, *p*=.07122. The second test found that more experience programmers used more operators to debug programs with non-compiler errors (*Mdn*=7.29) than novices (*Mdn*=6.96), *W*=5220.5, adj. α=.025, *p*=.01845. A separate Mann-Whitney found more experienced (*Mdn*=1.03) and novice programmers (*Mdn*=1.04; *p*=.3852) did not differ in the use of evaluations on failed programs with the automated feedback tool (*p*>.05).

Which COPES features are important for classifying if students learned?

For RQ 2, a random forest classifier was built to predict whether a student learned, from mid- to post-test using COPES variables. For that classifier, we split students into a group that learned (NLG>0) and a group that did not learn (NLG=<0). Data were partitioned into a 75:25 split for training and testing. To strike a balance between the best AUC and the lowest

error rate, grid search was used for the 1) number of trees grown and 2) COPES features used (Table 1) to grow each tree. Ten repetitions of 10-fold cross-validation (in the training) were used. The random forest classifier achieved an AUC ROC of .63 (95% CI [.54-.72] and Cohen's kappa of .2553, with 2 features per decision tree. The important features for classifying NLG were, first, conditions, general programming prior knowledge, and second, perceived programming ability. The third most important feature was the evaluations students used during programming, followed by the operators used to debug non-compiler errors. The operators used to debug compiler errors had a mean importance of 0.

4. Discussion

Novice programmers do not utilize SRL effectively with automated feedback tools in CS education (Ko et al., 2019; Loksa et al., 2022; Marwan et al., 2022). In this paper, we explored the role of prior knowledge on SRL and its relation to learning outcomes. The first RQ found that novices used operators more to debug compiler errors on failed programs. In contrast, more experienced programmers used more operators to debug non-compiler errors. By contrast, the groups did not differ in the evaluations of failed programs.

The second RQ found that the most important COPES feature for predicting students' learning gain was their level of general programming knowledge prior to the course. This was followed by their perceived programming ability, how often they evaluated failed programs with the automated feedback tool, and use of operators to debug non-compiler errors. The number of operators used to debug compiler errors was not part of the predictors of students' learning. Although syntax plays important role at the initial stage, its importance decreases over time. This work advances our understanding on the role of prior programming knowledge on SRL with an automated feedback tool and its relation on learning outcomes in CS education. This research has two limitations. First, the use of log data to study COPES, possibly missing other COPES behaviors that occur beyond system interactions, and 2) products were not collected, possibly missing additional data on SRL during programming.

References

Arakawa, K., Hao, Q., Greer, T., Ding, L., Hundhausen, C. D., & Peterson, A. (2021). In situ identification of student self-regulated learning struggles in programming assignments. In *Proceedings of the 2021 ACM Technical Symposium on Computer Science Education* (pp. 467-473). ACM.

Chen, H. M., Nguyen, B. A., Yan, Y. X., & Dow, C. R. (2020). Analysis of learning behavior in an automated programming assessment environment: A code quality perspective. *IEEE Access*, *8*, 167341-167354.

Ko, A. J., LaToza, T. D., Hull, S., Ko, E. A., Kwok, W., Quichocho, J., ... Pandit, R. (2019). Teaching explicit programming strategies to adolescents. In *Proceedings of the 2019 ACM technical symposium on computer science education* (pp. 469-475). ACM.

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).

Marwan, S., Akram, B., Barnes, T., & Price, T. W. (2022). Adaptive immediate feedback for block-based programming: Design and evaluation. *IEEE Transactions on Learning Technologies*, *15*(3), 406-420.

Marx, J. D., & Cummings, K. (2007). Normalized change. *American Journal of Physics*, *75*(1), 87–91.

Pankiewicz, M. (2020). Move in the right direction: Impacting students' engagement with gamification in a programming course. In *EdMedia+ Innovate Learning* (pp. 1180-1185). AACE.

Prather, J., Becker, B. A., Craig, M., Denny, P., Loksa, D., & Margulieux, L. (2020). What do we think we think we are doing? Metacognition and self-regulation in programming. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (pp. 2-13).

Winne, P. H., & Hadwin, A. F. (1998). Studying as self-regulated learning. In D. J. Hacker, J. Dunloksky, & A. C. Graesser (Eds.), *Metacognition in educational theory and practice* (pp. 277-304). Erlbaum.