Identifying and Evaluating Novel Knowledge Component Models for Programming Skills

Mehmet Arif Demirtaş
University of Illinois Urbana-Champaign
Urbana, IL, USA
mad16@illinois.edu

ABSTRACT

Identifying key concepts in programming is important for accurately tracking skill development and designing better support mechanisms for students. Prior research has identified a plethora of skills at varying levels of granularity, from broad abilities such as code comprehension and tracing to fine-grained skills like using individual syntactic elements correctly. However, the extent to which these skill models reflect students' skill acquisition process has not been empirically explored. Knowledge components, which are acquired units of skills and abilities inferred from the performance on a set of tasks, can be an appropriate starting point in a framework for evaluating these skill models and generating new models in a data-driven manner. Yet, previous applications of knowledge components for programming could not identify a robust and interpretable skill model, which may imply that programming skills are more complex than initially assumed. Thus, we aim to design a knowledge component model for programming that reflects student development while maintaining the interpretability of individual components. My findings so far have shown that language elements, combined with the context they are used in, can viably model the skill acquisition process of students. In my future work, I aim to design tools for identifying the combinations of language structures that accurately reflect the skill acquisition process in programming.

Keywords

computing education research, knowledge components, programming plans

1. INTRODUCTION & BACKGROUND

Identifying the skills acquired while learning to program is an important goal in computer science education. Understanding these skills in depth can help in tracking student development accurately, as well as designing effective learning environments that better support novice learners. However, the computing education community lacks an evidence-based model for fine-grained skills developed during the learning of programming. Abilities such as code comprehension and writing syntactically correct code have been identified to be crucial in introductory programming courses [24], but the breadth of these abilities makes it more difficult for these to be used for knowledge tracing and personalized support mechanisms. Other works surveyed programming courses to identify atomic skills like using individual syntactic elements (e.g. operators, conditional structures, or types of loops) as fundamental concepts in introductory computing [23]. Still, these concepts were not based on student data but rather reflected expert consensus, which can be susceptible to biases like expert blind spot [15]. Many EDM studies for understanding student skill development on programming data have also leveraged domain expertise [8, 19, 2, 21].

Knowledge components (KC), acquired units of skill that can be detected by measuring student performance on a set of tasks, can provide a useful framework for identifying key skills in a data-driven manner. Originally defined as part of the Knowledge-Learning-Instruction (KLI) framework [12], knowledge components can be used for both tracking student learning and modeling the skills that are being learned. According to the KLI framework, students should achieve lower error rates in problems related to a knowledge component as they get more opportunities to practice the underlying skill. Thus, a knowledge component model can be evaluated on student data using learning curves, by tracking the change in error rate as students practice the skill represented by a KC [6]. Learning curve analysis has been successfully applied for tracking skill development and refining skill models in many domains through knowledge components, including algebra, physics, geometry, and language studies [11, 13, 16].

In contrast to these other fields that utilized knowledge components for improving skill models, previous applications with KCs in programming have yielded mixed results. Early studies have successfully utilized manually annotated KCs on problems for supporting students e.g. through individualized problem selection [4]. However, this approach was not scalable to open-ended problems as it required manual annotation. Rivers et al. [19] was the first work to evaluate the extent to which language features can model student skill acquisition in programming education by using abstract syntax tree (AST) nodes as a KC model on open-ended code-writing data. Surprisingly, their learning curves did not show a decrease in error rate for many of the AST nodes (e.g. for loops, return statements, addition operation), leading to

M. A. Demirtas. Identifying and evaluating novel knowledge component models for programming skills. In B. Paaßen and C. D. Epp, editors, *Proceedings of the 17th International Conference on Educational Data Mining*, pages 969–973, Atlanta, Georgia, USA, July 2024. International Educational Data Mining Society.

https://doi.org/10.5281/zenodo.12730015

questions on the validity of the use of syntax elements as an accurate model of skills in programming. Shi et al. [21] used an alternative approach to develop a KC model from open-ended data by optimizing a neural network to predict sets of problems that generate successful learning curves, and manually inferring KCs through interpretation of the problems included in the set at the end of this optimization process. However, their work was limited in interpretability as the identified KC candidates could combine multiple unrelated concepts observed in student submissions as a single KC. These studies may imply that code-writing data collected from open-ended submissions for programming exercises, a common form of data collected in programming courses, is not appropriate for learning curve analysis. However, our initial findings imply otherwise.

Our preliminary evidence suggests that learning curve analysis can identify programming KCs under certain conditions. In our recent work, we replicated Rivers et al.'s study [19] on modeling student learning with syntactic elements with a larger dataset and showed that AST nodes can be a viable KC model, contrary to their findings. The identified KC model had a better model fit as measured by AIC, BIC, and RMSE [13], and the individual KCs produced learning curves with steady declines in error rates, indicating mastery of the underlying skills [12]. However, we also observed that AST nodes that better represent learning tend to be used in similar contexts in most programs, such as conditional statements and loops, and AST nodes for language elements used in many different places throughout the program, such as operators, had poor performances as KCs.

Inspired by this finding, we intend to develop a methodology for identifying combinations of language structures that accurately model student skill acquisition in a data-driven manner. We believe that programming plans and common code patterns can be useful candidates as structures that capture underlying skills. Previous literature on plans and patterns identified these structures through more subjective methods, such as surveying experts or conducting cognitive task analyses, which may not necessarily reflect the skill acquisition process of novices at large. By using KCs as a framework for evaluating skill models, we aim to identify *evidence-based programming plans* that can be validated on student data, using methods such as learning curve analysis. The primary research questions of interest are as follows:

1. RQ1: To what extent does students' ability to use syntactic elements model skill acquisition in programming?
2. RQ2: To what extent does students' ability to use programming plans model skill acquisition in programming?
3. RQ3: How can we develop and evaluate novel knowledge components that model skill acquisition in programming?

Section 2.1 details my previous study and findings addressing RQ1. Section 2.2 explains the design and the preliminary results from a new study that addresses RQ2 and Section 2.3 discusses some possible approaches for answering RQ3.

2. RESEARCH PROGRESS

2.1 Evaluating syntax elements as a skill model

In our recent work, we have shown that the ability to use syntactic elements can model skill acquisition in introductory programming to some extent contrary to prior findings. Our work was a replication of a study by Rivers et al. [19] on a larger dataset collected over seven semesters of instruction in an introductory programming course, including significantly more programming exercises compared to previous studies. Using the knowledge component model based on abstract syntax tree (AST) nodes proposed in the previous study [19], we compared ASTs generated from student submissions and correct solutions to detect which AST nodes are used correctly in a student submission. By treating each AST node as a separate knowledge component, each student submission provided an attempt at many KCs at once to be analyzed using learning curve analysis (Figure 2). While the prior studies using this KC model achieved learning curves that do not show any clear learning trend [19, 21], our learning curves showed a steady decline in error rates, indicating healthy learning across many knowledge components. Moreover, our knowledge component model had a better model fit and higher predictive power compared to baselines from previous studies.

In our further analysis, we noted that AST nodes that represent control flow structures ("If", "For", "Return") and data structure declarations ("Dict", "Tuple") exhibited a more consistent decline in error rate across semesters, compared to nodes for operators. Two example learning curves are shown in Figure 1. We attribute this difference to some language structures appearing in mostly similar places and contexts in correct solutions: in many of our exercises, data structure declarations would follow the function signature, and control flow structures would be at the end of the program. Thus, students can start to correctly place these structures after a few attempts. On the other hand, operators could appear anywhere in the program, making it more difficult for students to infer which conditions create the required context for the use of a specific operator.

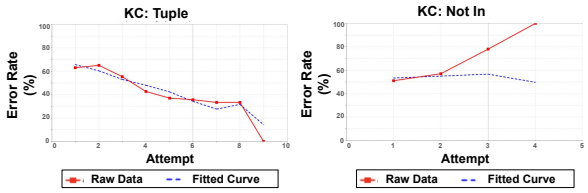

Figure 1: An example learning curve for two knowledge components from the same semester. The curve for "Tuple" shows that as students have more attempts at practicing this KC, error rates at problems steadily decrease. On the other hand, the curve for the "NotIn" operator does not show a declining trend in observed attempts.

2.2 Evaluating programming plans as a skill model

Inspired by this finding, we adopted programming plans [22] as an alternative to syntactic features as a skill model. We used programming plans as KCs on the same dataset described above to understand if programming plans can model

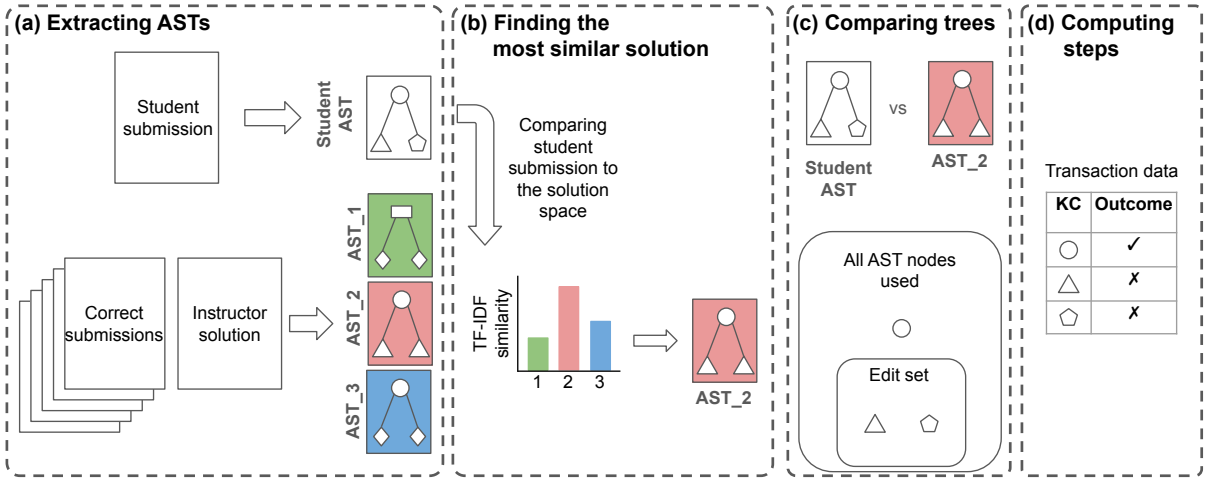

Figure 2: Our pipeline for extracting AST-based knowledge components from student submissions (Section 2.1). For each student submission, the correct solution that is syntactically most similar to the submission is found and used as ground truth to identify which KCs are correctly used in the given submission. See original work for details.

```
def count_over_100(number_list):
    count = 0
    for num in number_list:
        if num > 100:
            count += 1
    return count
```

```
def count_string_vals(a_dict):
    count = 0
    for val in a_dict.values():
        if type(val) == str:
            count +=1
    return count
```

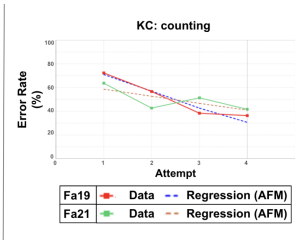

Figure 3: Two examples of the counting plan, along with learning curves from two semesters, showing a decrease in error rates.

skill acquisition better than language elements. To this end, we annotated a set of more than 300 programming exercises with programming plans proposed by Iyer et al. [9]. We consider student submissions on these problems as attempts made to practice the KCs represented by the annotated plans.

By testing programming plans on student data, we produce empirical evidence to support the validity of programming plans as a mental construct as well as evidence on which plans correspond to discrete skills learned by students. Our results show that some plans (e.g., *counting*, *sum*) can model student learning in introductory programming well, while others do not. Figure 3 shows the two examples with the *counting* plan and learning curves from two semesters on the plan. Interestingly, the plans that had consistently successful learning curves were plans that had more specific use cases, whereas plans that only described a common structure (e.g. multiway branching) had learning curves with no consistent decrease in error rates.

However, we also observed that manually annotating plans only on a problem level presents an obstacle to collecting accurate data on some plans, pointing to a need for an automatic method for detecting these structures in submissions.

2.3 Generating new knowledge components for programming

Influenced by the gaps we have identified so far, we aim to develop a data-driven methodology for identifying structures that reflect skills acquired in programming courses. While our studies showed promising results for both syntactic elements and programming plans, a method for identifying structures that optimize the model fit on learning curves, similar to Shi et al.'s work [21], can improve our knowledge component models. Thus, I plan to examine optimization-based processes for refining a proposed set of knowledge components, which can help in finding a balance between model fit and interpretability.

Moreover, with the capabilities of large language models (LLM) in computing education becoming ubiquitous [18], processing student submissions with LLMs can be an alternative approach for identifying common patterns. The capabilities of these models in explaining code, generating code pieces, and programs with explanations have been discussed [20, 3, 10]. Knowledge components proposed by LLMs conditioned on student submissions can contribute to this discussion by exploring the connection between generated code pieces and the skill acquisition process of students.

Another possible approach could be merging multiple language elements, such as combining the use of multiple AST tokens as a single node as proposed in the discussion by Rivers et al. [19]. Based on the findings from our replication, combining multiple AST nodes to add context information to knowledge components may result in a domain model with better represents student learning. Learning Factors Analysis (LFA) [6], a skill model improvement method guided by model fit metrics, provides a possible tool for this exploration.

3. CONCLUSION

Generating a skill model in programming that can capture the use of combinations of syntactic elements, organized

in a certain way under a particular context, may model student learning accurately while also being easily interpretable. With automatic detection of such interpretable structures, it can be possible to track student learning better through knowledge tracing techniques [1]. A fine-grained skill model represented by these structures can also simplify the implementation of scaffolding techniques such as subgoal labeling [5, 14] or Parsons problems [17, 7] in any domain with a sufficiently large set of example programs. During the doctoral consortium, I hope to discuss possible methods for generating these skill models and designing studies that can better evaluate the generated skill models.

4. REFERENCES

[1] G. Abdelrahman, Q. Wang, and B. Nunes. Knowledge Tracing: A Survey. *ACM Computing Surveys*, 55(11):224:1–224:37, Feb. 2023.

[2] B. Akram, H. Azizolsoltani, W. Min, E. N. Wiebe, A. Navied, B. W. Mott, K. E. Boyer, and J. C. Lester. A Data-Driven Approach to Automatically Assessing Concept-Level CS Competencies Based on Student Programs. In *CSEDM@ EDM*, 2020.

[3] B. Akram and A. Magooda. Analysis of Students' Problem-Solving Behavior when Using Copilot for Open-Ended Programming Projects. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 2*, volume 2 of *ICER '23*, page 32, New York, NY, USA, Sept. 2023. Association for Computing Machinery.

[4] V. Aleven and K. R. Koedinger. Knowledge component (KC) approaches to learner modeling. *Design recommendations for intelligent tutoring systems*, 1:165–182, 2013.

[5] R. Catrambone. The subgoal learning model: Creating better examples so that students can solve novel problems. *Journal of experimental psychology: General*, 127(4):355, 1998.

[6] H. Cen, K. Koedinger, and B. Junker. Learning Factors Analysis – A General Method for Cognitive Model Evaluation and Improvement. In M. Ikeda, K. D. Ashley, and T.-W. Chan, editors, *Intelligent Tutoring Systems*, Lecture Notes in Computer Science, pages 164–175, Berlin, Heidelberg, 2006. Springer.

[7] B. J. Ericson, P. Denny, J. Prather, R. Duran, A. Hellas, J. Leinonen, C. S. Miller, B. B. Morrison, J. L. Pearce, and S. H. Rodger. Parsons Problems and Beyond: Systematic Literature Review and Empirical Study Designs. In *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '22, pages 191–234, New York, NY, USA, Dec. 2022. Association for Computing Machinery.

[8] C. Fernandez-Medina, J. R. Pérez-Pérez, V. M. Álvarez-García, and M. d. P. Paule-Ruiz. Assistance in computer programming learning using educational data mining and learning analytics. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 237–242, New York, NY, USA, July 2013. Association for Computing Machinery.

[9] V. Iyer and C. Zilles. Pattern Census: A Characterization of Pattern Usage in Early Programming Courses. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, SIGCSE '21, pages 45–51, New York, NY, USA, Mar. 2021. Association for Computing Machinery.

[10] B. Jury, A. Lorusso, J. Leinonen, P. Denny, and A. Luxton-Reilly. Evaluating LLM-generated Worked Examples in an Introductory Programming Course. In *Proceedings of the 26th Australasian Computing Education Conference*, ACE '24, pages 77–86, New York, NY, USA, Jan. 2024. Association for Computing Machinery.

[11] K. Koedinger, K. Cunningham, A. Skogsholm, and B. Leber. An open repository and analysis tools for fine-grained, longitudinal learner data. In *Educational Data Mining 2008*. Citeseer, 2008.

[12] K. R. Koedinger, A. T. Corbett, and C. Perfetti. The Knowledge-Learning-Instruction Framework: Bridging the Science-Practice Chasm to Enhance Robust Student Learning. *Cognitive Science*, 36(5):757–798, 2012.

[13] K. R. Koedinger, E. A. McLaughlin, and J. C. Stamper. Automated Student Model Improvement. Technical report, International Educational Data Mining Society, June 2012.

[14] B. B. Morrison, L. E. Margulieux, B. Ericson, and M. Guzdial. Subgoals Help Students Solve Parsons Problems. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 42–47, New York, NY, USA, Feb. 2016. Association for Computing Machinery.

[15] M. J. Nathan, K. R. Koedinger, and M. W. Alibali. Expert blind spot: When content knowledge eclipses pedagogical content knowledge. In *Proceedings of the Third International Conference on Cognitive Science*, volume 644648, pages 644–648, 2001.

[16] H. Nguyen, Y. Wang, J. Stamper, and B. M. McLaren. Using Knowledge Component Modeling to Increase Domain Understanding in a Digital Learning Game. Technical report, International Educational Data Mining Society, July 2019.

[17] D. Parsons and P. Haden. Parson's programming puzzles: A fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*, pages 157–163, 2006.

[18] J. Prather, P. Denny, J. Leinonen, B. A. Becker, I. Albluwi, M. Craig, H. Keuning, N. Kiesler, T. Kohn, A. Luxton-Reilly, S. MacNeil, A. Petersen, R. Pettit, B. N. Reeves, and J. Savelka. The Robots Are Here: Navigating the Generative AI Revolution in Computing Education. In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '23, pages 108–159, New York, NY, USA, Dec. 2023. Association for Computing Machinery.

[19] K. Rivers, E. Harpstead, and K. Koedinger. Learning Curve Analysis for Programming: Which Concepts do Students Struggle With? In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ICER '16, pages 143–151, New York, NY, USA, Aug. 2016. Association for

Computing Machinery.

[20] S. Sarsa, P. Denny, A. Hellas, and J. Leinonen. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1*, volume 1 of *ICER '22*, pages 27–43, New York, NY, USA, Aug. 2022. Association for Computing Machinery.

[21] Y. Shi, R. Schmucker, M. Chi, T. Barnes, and T. Price. KC-Finder: Automated Knowledge Component Discovery for Programming Problems. Technical report, International Educational Data Mining Society, 2023.

[22] E. M. Soloway and B. Woolf. Problems, plans, and programs. *ACM SIGCSE Bulletin*, 12(1):16–24, Feb. 1980.

[23] A. E. Tew and M. Guzdial. Developing a validated assessment of fundamental CS1 concepts. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, pages 97–101, New York, NY, USA, Mar. 2010. Association for Computing Machinery.

[24] B. Xie, D. Loksa, G. L. Nelson, M. J. Davidson, D. Dong, H. Kwik, A. H. Tan, L. Hwa, M. Li, and A. J. Ko. A theory of instruction for introductory programming skills. *Computer Science Education*, 29(2-3):205–253, July 2019.