Board 242: Developing Valid and Equitable Tasks for Assessing Programming Proficiency: Linking Process Data to Assessment Characteristics

Dr. Mo Zhang, Educational Testing Service
Amy Jensen Ko, University of Washington
CHEN Li, Educational Testing Service

Developing Valid and Equitable Tasks for Assessing Programming Proficiency: Linking Process Data to Assessment Characteristics

Mo Zhang
mzhang@ets.org
Educational Testing Service

Amy Ko
ajko@uw.edu
University of Washington

Min Li
minli@uw.edu
University of Washington

Hongwen Guo
hguo@ets.org
Educational Testing Service

Ben Zhou
benzhou@uw.edu
University of Washington

Chen Li
cli@ets.org
Educational Testing Service

Abstract

Learning to code is becoming a popular subject for students and professionals of all ages, partly for its career prospects, but also as a critical literacy for understanding how computing is shaping society. Yet, educators generally agree that computer programming is difficult to teach and assess. This poster presents an in-progress research that aims to address difficulties in assessing computer programming by investigating critical characteristics of programming tasks using both response process and product data.

Introduction

Learning to code is becoming a popular subject for students and professionals of all ages, partly for its career prospects, but also as a critical literacy for understanding how computing is shaping society. Yet, educators generally agree that computer programming is difficult to teach and assess. This poster presents an in-progress research study that aims to address difficulties in assessing computer programming by investigating critical characteristics of programming tasks using both response process (e.g., key presses, mouse clicks) and product data (i.e., submitted codes).

Despite the progress made in understanding the difficulties in learning programming and in instructional interventions, there has been little progress in assessing programming skills. The assessment chapter by Fincher and Robins[1] in a recent book surveying the entire field of Computer Science education highlights this gap by noting a severe lack of research on designing

valid, reliable, and fair measures of programming knowledge. Some recent work includes applying advanced psychometric methods to evaluate CS concept inventories[2] and applying modern validity frameworks[3] to assess specific skills[4]. While this progress is meaningful, prior work has overlooked the tensions between assessing the final program that a learner produces versus the process that a learner followed to produce that program. An open challenge in CS assessment and education is understanding the relationship between a learner's process and the programs resulting from that process[5].

Research on student's programming process mostly looks at consecutive snapshots of programs under construction[6,7]. Akram et al.[8] and Miao et al.[9] identified snapshot features that could be incorporated into constructive feedback to students as they are coding. While these studies illustrate potential uses of programming process data, none of the existing literature is focused on using that data to assess programming skills directly. This research addresses this gap by explicitly analyzing process data to explore how student programming strategies and processes are affected by programming task complexity and characteristics.

In this National Science Foundation sponsored collaborative research, we leverage the capability to record students' programming process using keystroke logs. One motivating factor for this research is based on the discovery and progress made in the area of keystroke log analysis in the domain of natural language writing[10]. We contend that learning and writing a programming language is also an act of writing, sharing similarities to writing in natural human language although with its own uniqueness. For example, just like writers, programmers often face an optimization problem: They must decide which goals to prioritize because they simply do not have the working memory to accomplish everything at once. In an assessment context, writing natural language and codes are potentially even more similar: both are done in response to a prompt that sets out expectations for the text to be produced, are evaluated according to specific criteria and, if the required text is sufficiently complex, students are likely to create plans for production that they then execute. Drawing from research on keystroke log analysis of natural language writing, we ask the research question: how do task complexity and characteristics relate to student programming process and performance? Fairness is a central concern of this research as well. To what extent do those task characteristics contribute to the performance patterns detected for students that vary along gender, ethnicity, and native language?

Methods

The first round of pilot data collection was recently completed with more than 170 students participated on a compensated, voluntary basis from universities in North America. Most students were enrolled in introductory Python programming courses at the time or prior to the participation. Students were asked to provide demographic background information in terms of their self-identified gender, ethnicity, and native language. All demographic questions were asked in an open-response format. Our research team developed 21 Python practice coding tasks with varying difficulty levels and targeted task characteristics. Each task included 7 or 8 test cases that students can run their programs against to check the correctness of their programs. The tasks were delivered from an online learning platform designed by researchers to facilitate research on programming language learning[11]. While students were allowed to attempt any single task as many times as they wanted, they were given two weeks in total to complete all tasks. Any

unfinished work was automatically logged by the system at the end of the 2-weeks window. All students, after they finished the programming tasks, were further invited to participate in a follow-up cognitive interview run by the research team. The cognitive interviews were implemented to validate some programming task design choices, process features extracted from keystroke logs, in particular, extended (long) pauses detected from the keystroke logs, as well as to provide substantive evidence on the cognitive processes underlying programming. There were 42 students who participated in the cognitive interviews on a voluntary basis.

Results

Comprehensive statistical analyses and process feature extraction from the keystroke logs are currently being conducted as of this writing. Preliminary qualitative analyses of the cognitive interview data suggested some revision of the item stem (i.e., the statement of the programming problem) to make the programming problems clearer to the students. Statistical and psychometric analyses on the response data, so far, revealed that expert-determined item difficulty can be largely confirmed by empirical evidence and only one programming task potentially favored the men group (i.e., the dominant student population in CS education) as opposed to the non-men group. Time spent on task analysis and efforts spent on code review, code revision or debugging have positive relations to programming proficiency. We are also developing a pipeline to extract meaningful process features from the keystroke logs. We plan to present analyses results at the conference as they become available in the near future.

Discussion

In summary, this project aims to advance our understanding of the cognitive processes underlying programming and inform ways to better teach, learn, and assess programming skills for all learners. We will illustrate an interdisciplinary approach to collecting, treating, and analyzing process data. That approach will help lead to a better understanding of how students' programming processes differ by their proficiency level and how processes interact with task characteristics.

References

[1] Sally A. Fincher and Anthony V. Robins. *The Cambridge handbook of computing education research*. Cambridge University Press, 2019.

[2] Benjamin Xie, Matthrew J. Davidson, Min Li, and Amy J. Ko. An item response theory evaluation of a language-independent cs1 knowledge assessment. In *SIGCSE '19: Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019.

[3] Michael Kane. Validating the interpretations and uses of test scores. *Journal of Educational Measurement*, 2013.

[4] Greg L. Nelson and Amy J. Ko. On use of theory in computing education research. In *Proceedings of 2018 International Computing Education Research Conference (ICER '18)*, 2018.

[5] Errol Thompson, Andrew Luxton-Reilly, Jacqueline L. Whalley, Minjie Hu, and Phil Robbins. Bloom's taxonomy for cs assessment. In *ACE '08: Proceedings of the tenth conference on Australasian computing education'*, 2008.

[6] Chris Piech, Mehran Sahami, Daphne Koller, Stephen Cooper, and Paulo Blikstein. Modeling how students learn to program. In *SIGCSE'12: Proceedings of the 43th ACM technical symposium on computer science education*, 2012.

[7] Paulo Bliktein, Marcelo Worsley, Chris Piech, Mehran Sahami, Steven Cooper, and Daphne Koller. Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences*, 2014.

[8] Bita Akram, Hamoon Azizolsoltani, Wookhee Min, Eric Wiebe, Anam Navied, Bradford Mott, Elizabeth B. Kristy, and James Lester. Automated assessment of computer science competencies from student programs with gaussian process regression. In *Proceedings of The 13th International Conference on Educational Data Mining (EDM 2020)*, 2020.

[9] Dezhuang Miao, Yu Dong, and Xuesong Lu. Pipe: Predicting logical programming errors in programming exercises. In *Proceedings of The 13th International Conference on Educational Data Mining (EDM 2020)*, 2020.

[10] Mo Zhang and Sandip Sinharay. Investigating the writing performance of educationally at-risk examinees using technology. *International Journal of Testing*, 2022.

[11] Benamin Xie, Jared O. Lim, Paul K. D. Pham, Min Li, and Amy J. Ko. Developing novice programmers' self-regulation skills with code replays. In *Proceedings of the 2023 ACM Conference on International Computing Education Research*, 2023.