Measuring Students' Programming Skill via Online Practice

Hongwen Guo[1], Mo Zhang[1], Amy J Ko[2], Min Li[2], Benjamin Zhou[2], Jared Lim[3], Paul Pham[2] and Chen Li[1]

[1]*ETS Research Institute, Princeton, New Jersey*
[2]*University of Washington, Seattle, Washington*
[3]*Georgia Tech University, Atlanta, Georgia*

Abstract

Allowing students to practice a programming task multiple times fosters resilience and improves learning. However, it is challenging to measure their programming skills in the dynamic and adaptive learning environment, in terms of determining the maximum allowed number of attempts, measuring progress during learning, and producing a fair performance score for different student groups. It is particularly challenging to do so when different students adaptively practice different sets of programming tasks. In this study, we leveraged data collected from an online learning platform in a pilot study and applied psychometric models to address two research questions: 1) How to measure students' progress in an adaptive practice setting that allows for multiple attempts and inform grading policies? and 2) How do different scoring rules affect bias analysis of the programming tasks? From the log data, we extracted two practice features (numbers of attempts and number of passed test cases), created six different scoring rules for scoring student's intermediate responses and final responses based on these practice features. We then used psychometric models and best practices to create a common scale to measure the dynamic performance that were comparable within individual students who made different attempts and across students who practiced different sets of programming tasks. This common scale ensured the comparability of performance within and across different student groups. It furthered enabled us to evaluate potential task biases between gender groups using the differential item functioning (DIF) analysis. Our preliminary results suggest that the final-attempt-based scoring rule not only boosted students' performance, but it also reduced the potential bias of programming tasks. This study contributes to methodologies in using log data to measure the dynamic programming skills and evaluate task biases in the adaptive online practice setting.

Keywords

Psychometrics, Python coding task, Scoring, Item Bias, Fairness

1. Introduction

As with any skill, practice makes perfect for programming skills in computer science (CS) education. Decades of research in learning theory has demonstrated the importance of deliberate practice and having a "coach" who provides feedback for ways of optimizing performance, no matter whether it is learning a new language, a new math topic, or a new programming skill [1]. With advances in technologies, many online learning platforms and computer science courses have been developed to align with the learning theory by allowing learners to practice a problem multiple times while providing immediate feedback, which encourage students to learn from their mistakes and strive for success [2]. A recent large-scale randomized control study [3] showed that students who used such a math learning platform learned significantly more, and the impact was greater for students with lower prior mathematics achievement. The benefit of timely feedback for programming assignments in CS education is also evident in a recent review [2].

However, repeated attempts in online practice pose challenges to assess students' skills. For example, how many attempts should be allowed? How to measure progress in practice? How can we generate fair scores for diverse student groups in CS learning contexts? These questions are especially important for assessing programming skills, since there are many different potential approaches to assessing students' performance when multiple attempts are allowed, while some methods may unintentionally further marginalize students from non-dominant groups in CS education [4, 5]. To examine the validity and fairness in CS assessments, a few recent research studies applied the established psychometric approaches such as the differential item functioning (DIF) analysis methods [6], but on students' final responses and fixed test format. For example, using data collected from a paper-and-pencil-based CS exam in a large CS1 course, Davidson and coauthors applied DIF methods to evaluate whether the tasks on the exam might potentially favor one student group over the other [7]. Xie and colleagues investigated potential bias in an online CS curriculum using DIF analysis and expert interpretations [8]. These studies highlighted the relevance of psychometrics and DIF analysis in the context of CS education.

It is particularly challenging to measure student's progress in an adaptive online learning environment (such as personalized learning), where students with different levels of skills are recommended to practice programming tasks with different levels of difficulties. The measurement needs to quantify both progress within a student and across student groups in such a learning environment.

In the current study, we framed our investigation with two preliminary two research questions to address the challenges: 1) How to measure students' progress in an adaptive practice setting that allows for multiple attempts and inform grading policies? and 2) How do different scoring rules affect bias analysis of the programming tasks?

To address the above two research questions, we investigate how rigorous educational measurement methods (i.e. statistical and psychometric models) can be used to create comparable performance scores that are comparable within and across students during practice. For this, it is necessary to create a common scale, which reflects the dynamic/intermediate programming processes with different numbers of attempts within individual students. It is critical as well that the common scale ensures that performance scores are comparable across students who practiced different sets of programming tasks. We created and examined six scor-

CSEDM'24: 8th Educational Data Mining in Computer Science Education Workshop, July 14, 2024, Atlanta, GA
hguo@ets.org (H. Guo); mzhang@ets.org (M. Zhang); ajk@uw.edu (A. J. Ko); minli@uw.edu (M. Li); benzhou@uw.edu (B. Zhou); jlim419@gatech.edu (J. Lim); pkdpham@uw.edu (P. Pham); cli@ets.org (C. Li)

ing rules based on different maximum numbers of allowed attempts. Under the different scoring rules, we applied rigorous statistical and psychometric methods [9] to create comparable performance scores, which further allowed for psychometric analysis of task biases [7, 10, 11, 8] between the men (the dominant CS student group) and the non-men groups.

The following sections provide a more in-depth description of our approaches. We first introduce the study design and data collection, and present our proposed methodologies to create a common scale for producing comparable performance scores and for detecting potential task bias. We conclude the study with a discussion section on our preliminary findings, the study limitations, as well as our future direction.

The current work explores an emerging terrain in adaptive learning and assessment in CS education. Our work is pioneering in integrating educational data mining (i.e., creation of programming practice features extracted from log data) and the innovative applications of psychometric models to build a common scale, measure processes, and evaluate potential task bias on online practice platforms.

2. Methodologies

This study was approved by an Institutional Review Board (IRB) prior to engagement with participants for data collection.

2.1. Study Design and Data Collection

The online learning platform we used for data collection is capable of providing immediate feedback on test cases (showing pass or fail) [12]. More importantly, in this study, the research team implemented an adaptive learning/assessment approach to recommend programming task sets that are suitable to students with different programming skills for effective learning and meaningful engagement [13].

To answer our research questions, we conducted a pilot study and recruited about college students on a compensated, volunteer basis from universities in North American who were enrolled in introductory Python programming courses currently or prior to participation. The participating students majored in different fields and varied in the number of programming related courses and years of experience in programming. For the bias analysis, we focus on self-described gender, which included categories such as men, women, non-binary, and free response options. In order to produce reliable bias analysis results, large samples are recommended; thus, we compromised on two student groups: men (N=91), the dominant gender group in computer science (CS) learning contexts who tend to be privileged in CS, and non-men (N = 63), including women, non-binary students, and students who reported other gender identities (exclude missing responses). This allowed us to study whether any programming task favored men over students with other gender identities, but did not allow us to do more fine-grained analysis in the pilot study.

Students practiced Python programming tasks on an online learning platform designed by researchers to facilitate research on programming language learning [12]. CS content experts on our team developed 21 Python coding tasks with 7 to 8 test cases per task. The tasks vary in their measurement specifics and difficulty level, which allowed for an

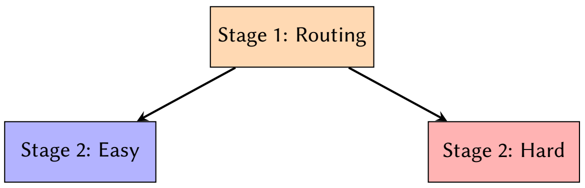


Figure 1: The Adaptive design with two stages and three item blocks

implementation of an adaptive two-stage test design (refer to Figure 1).

This adaptive design has been gaining attention and popularity in learning and, particularly assessment field, mainly because it takes into account the differences in student skill levels and provides different tasks or different sets of tasks to suit students' skill levels ([14, 15, 13]). All students answered a common task set of medium difficulty at the first stage and were routed to either an easy or a hard task set at the second stage depending on how well they did in the first stage (refer to Figure 1). Based their final submitted responses on the common task set, students who scored in the lower half of the score distribution were routed to the easy block, those who scored in the upper half were routed to the hard task set. The task sets at the second stage were more aligned to students' skill levels, and hopefully students would be more engaged with the practice.

The task delivery platform provides students with immediate feedback on which test cases they passed. Students were allowed to attempt a task as many times as they like, or until they passed all the test cases. Students were given a 2-week window to complete all the tasks, and were given \$80 in recognition of their time and effort upon completing the tasks. Students who invested a mean of less than 3 minutes of effort into each problem set were not compensated as this indicated either no effort to seek correct solutions or use of external aids to generate solutions. This was roughly 5% of the initial set of the participants, and some of these participants confirmed that they were just seeking compensation.

2.2. Data Preparation

2.2.1. Practice Features

Log data were collected from the online platform, which contained fine grained information on what codes students produced, what actions they took, and for how long, etc.. The fine-grained data have shown to be very useful in understanding students' problem-solving processes and performance [16]. In this preliminary study, we focused on two practice features that capture students' intermediate steps before they submitted their final codes: the number of attempts (i.e., number of running/testing their codes) and how many test cased were passed in each attempt (please also refer to Table 1 below for coding rules). More comprehensive analysis of the log data will be conducted in further studies and in the next phase when larger samples are collected. Note that in the following, a programming task is also referred to as an item in psychometric modeling.

Table 1
Six scoring rules based on different number of attempts on a task.

S1:	$x = 1$ if $E = True$ in the 1st attempt, otherwise 0.
S2:	$x = 1$ if $E = True$ in the 2nd attempt, otherwise 0. If they did not have the 2nd attempt, S2=S1.
S3:	$x = 1$ if $E = True$ in the 3rd attempt, otherwise 0. If they did not have the 3rd attempt, S3=S2.
S4:	$x = 1$ if $E = True$ in the 4th attempt, otherwise 0. If they did not have the 4th attempt, S4=S3.
S5:	$x = 1$ if $E = True$ in the 5th attempt, otherwise 0. If they did not have the 5th attempt, S5=S4.
SF:	$x = 1$ if $E = True$ in their last attempt, otherwise 0.

2.2.2. Different scoring rules

In preparing the data to create a common scale, we dichotomously scored each programming task for each student in each attempt. This choice was constrained by the two facts: the sample size in the pilot study was relatively small, which could only support the use of the simplest psychometric model with dichotomouse responses (i.e, Rasch model; [6]), and the empirical data showed that the numbers of passed cases concentrated in the two ends (either a very low number or a very high number). Please refer to the Results section and Figure 3. In this study, we experimented with six scoring/grading rules based on the process features. Table 1 provides a detailed description for each scoring rule on an item, where x stands for the item score of a student, and E for the event that the student passed at least half of the test cases.

2.3. Data Modeling

In this section, we provide concise and conceptual descriptions of the used psychometric and statistical methods. Interested readers are recommended to refer to the cited papers for technique details of these methods.

2.3.1. Item Response Models

Because of the adaptive study design, there was missing data by design. The total sum of passed cases of all tasks is not comparable across students as some students took the harder task set and some took the easier one. It was necessary to create a common (base) scale for comparability. We applied the Item Response Theory (IRT) models [6] in psychometrics that uses information from the observed task responses for task difficulty calibration (i.e., estimation). Due to the relatively small sample size (N = 159) we applied Rasch model, the IRT model with the least number of parameters, to obtain reliable parameter estimates [6]. In a Rasch model (i.e., a latent logistic regression model), the probability for Student i to get a correct answer on Task j (i.e., $x_j = 1$) depends on the student's latent ability θ_i and the item difficulty d_j:

$$P(x_j = 1|\theta_i, d_j) = \frac{1}{1 + \exp\{-(\theta_i + d_j)\}}. \quad (1)$$

A student with higher programming proficiency (i.e., larger θ) has a higher probability of getting the item correct, but a harder item (indicated by a lower value of d) decreases that chance. Note that in the following, to be consistent with psychometric terminology, a programming task is also referred to as an item.

Item Set	Sample	Item difficulty
Common set	Every Student	d_{common}
Easy set	Lower half scorers	d_{easy}
Hard set	Upper half scorers	d_{hard}

Table 2
Item calibration samples

2.3.2. Item calibrations

Because the sample sizes were relatively small, and they were even smaller in the second stage of the study design, we used a two-step approach to conduct item calibrations. There are three sets of items and associated item difficulty estimates (refer to Table 2 below).

In the first step, we used Rasch model to estimate d_{common} for items in the common item set that everyone practiced at stage one; in the second step, we used the fix-parameter calibration method [17, 18] to obtain d_{easy} and d_{hard} estimates for items at stage two in the easier set and hard set, respectively. In the second step, d_{common} estimates were fixed, so that all ds were on the same scale. The two-step item calibration approach can help to obtain more reliable and accurate estimates of parameters ds (refer to [18] and references therein).

Note that the item calibrations were conducted on the first attempt only; that is, items were scored using S1 in Table 1. These item difficulty parameters (ds) were used as our common (based) scale for the subsequent analyses, which ensured that all subsequent performance scores based on different scoring rules were comparable.

2.4. Performance Scores

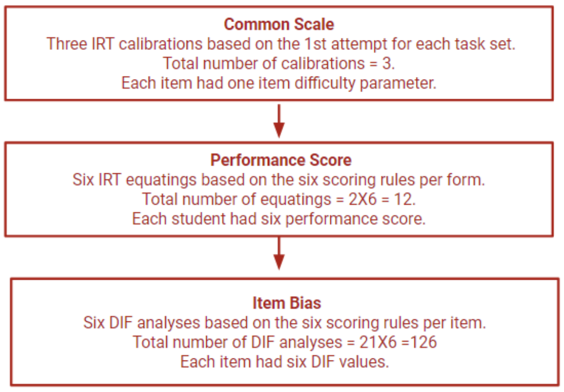
For a test form consisting of $J = 21$ items, the true score T_i [9] for Student i is defined as

$$T_i = \sum_{j=1}^{J} P(x_j = 1|\theta_i, d_j) = \sum_{j=1}^{J} \frac{1}{1 + \exp\{-(\theta_i + d_j)\}}. \quad (2)$$

However, again, because different students took different test forms, either Form 1 (common item set plus the easy item set; $J_1 = 15$), or Form 2 (common item set plus the hard item set; $J_2 = 15$), the true score on different test forms are not comparable either. Thus, it is necessary to "equate" true scores from one form to the other form to produce comparable scores. The equating process was realized through the IRT equating procedure [9] to produce an equated score for each student through the common scale determined by item parameters ds obtained in the above item calibration steps.

In our study, for each of the six scoring rules, two IRT equatings were conducted: one from Form 1 to Full Form (the full set; J=21), and the other from Form 2 to Full Form. As such, the IRT equating acted as an imputing method, and the equated score was set on Full Form to reflect a student's true score as if the student took the full set of 21 items.

Applying the six different scoring rules in Table 1 for item score x in Equation 1, we produced six equated scores for each student. Because item parameters ds were the base scale and used in all IRT equatings, the equated scores were comparable across students and scoring rules. The equated scores can, therefore, be used as performance scores for comparison and the changes in these performance scores (based on different scoring rules) reflect skill progress during the programming processes on the tasks.



Common Scale
Three IRT calibrations based on the 1st attempt for each task set.
Total number of calibrations = 3.
Each item had one item difficulty parameter.

Performance Score
Six IRT equatings based on the six scoring rules per form.
Total number of equatings = 2X6 = 12.
Each student had six performance score.

Item Bias
Six DIF analyses based on the six scoring rules per item.
Total number of DIF analyses = 21X6 =126
Each item had six DIF values.

Figure 2: Architecture

2.5. Item Bias Detection

It is sensible to evaluate biases at the item level since performance score is a function of aggregated item scores (using the psychometric models as described above). For each item, we conducted differential item functioning (DIF) analyses using the average item scores between the non-men (focal) group to the men (reference) group [19]. There are various statistical approaches for DIF analyses [6, 19]. In this study, we applied a DIF approach that can evaluate the difference in the average item scores (standardized mean difference, SMD), as well as the difference in the average attempt (differential number of attempts), conditioning on students who have similar programming skills.

More specifically, let X be the item score (0 or 1), and T be the total performance score [20]. To assess whether an item functioned differently for students in two different groups, a studied/focal group (Group f) and a comparison/reference group (Group r), comparison is made between the expected item scores for given total scores, $E_f(X|T)$ and $E_r(X|T)$. The SMD is a weighted sum of the differences of conditional expectations between the focal and reference groups for an item; that is

$$\text{SMD} = \sum_t w_{ft}[E_f(X|T=t) - E_r(X|T=t)],$$

where w_{ft} is the proportion of the focal group members in the t-score group. In practice, $E(X|T)$ is estimated by the average item scores in the t-score group. In the DIF context, T is often called the matching variable [11, 10, 20].

A statistically significant and negative SMD may indicate that the studied item favors the reference group, and a statistically significant and positive one the focal group. The choice of this DIF method was based on the small sample sizes in our pilot data and the intuitive interpretation of the DIF effect size (i.e. SMD). Similarly, differential number of attempts is the difference in the weighted average attempt numbers between the two groups after matching on the performance score.

Figure 2 summarizes the architecture of modeling and analyses used in the study.

In the study, we used the R package *mirt* [21] to conduct IRT item calibrations and programmed R codes to run the other statistical and psychometric analysis. Interested readers are referred to the references cited for technical details. Some useful R codes for item calibration and score equating are also available in [18].

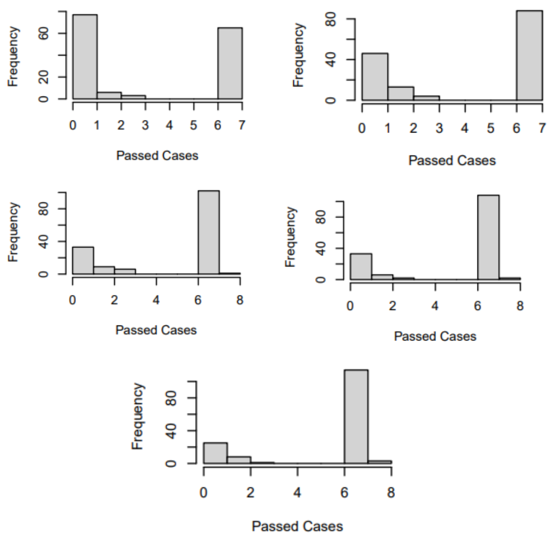


Figure 3: Distribution of numbers of passed cases on one item. In each panel from left-to-right and top-to-bottom, the plot shows the frequency Distribution of passed cases based on the first one, first two, first three, first four and first five attempts, respectively, on one task that had eight test cases. Note that in the first two attempts, no students got all 8 cases correct.

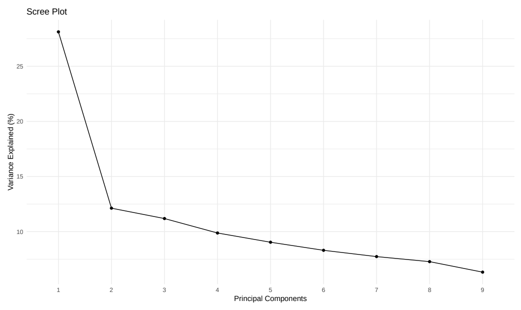


Figure 4: Dimensionality Analysis

3. Results

3.1. Data Summary

As discussed earlier, if students kept trying a programming task, they were most likely to pass all the test cases. Figure 3 shows the typical data pattern on one task. Most students' numbers of passed cases were either close to 1 or close to 7, and the counts around 1 shifted to 7 or 8 as they made more attempts on the task. The same pattern was observed on most of the tasks, and thus they supported the choice of scoring the tasks dichotomously.

The preliminary PCA analysis on the common item set showed that the dichotomously scored responses had one dominant dimension (i.e., uni-dimensionality was acceptable. Refer to Figure 4).

3.2. Item Difficulty Distribution

Figure 5 presents item difficulty, calibrated on the first attempt (S1) from the Rasch model. It was observed that item difficulty of the 21 programming tasks had a good spread: Tasks 8, 19, 20 and 21 were challenging to the participants, Task 10 was very easy, and the rest tasks were somewhere

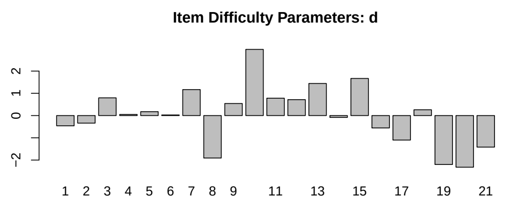


Figure 5: Item difficulty (*d*) estimates of the 21 tasks based on the first scoring rule (S1). A positive bar indicates that the item was less difficult for the non-men group.

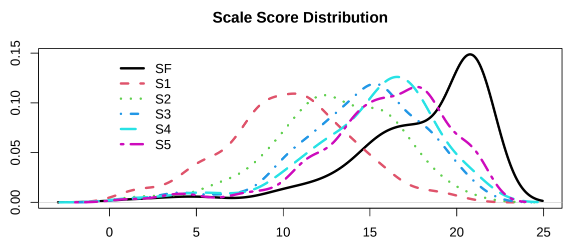


Figure 6: The six score distributions produced by the six different scoring rules, respectively, from S1 to S5, and to SF. The distributions shifted to higher scores as more attempts were allowed.

in between.

3.3. Performance Score Distributions

Performance score distributions based on different scoring rules (S1 to S5, and SF as in Table 1) are shown in Figure 6. As expected, as students practiced more on the programming tasks, they made progresses and their performance became better (except for a few students on the left tails of the distributions who were likely to give up early). Overall, students' performance scores based on the final attempts (SF) are clearly better than the other intermediate scores (the solid black curve as shown in Figure 6).

3.4. Potential Item Bias

Item bias analysis results show that there was an overall trend of reduced bias as students attempted more times on the tasks. Using an effect size of 0.1 as a threshold, the commonly used cut point to flag a meaningful difference in SMD (refer to [11] and references therein), Figure 7 shows that, in the first attempts (denoted as the solid orange circles), Items 12 and 14 were easier, but Items 2, 11, 16 and 21 were harder for the non-men group. However, the magnitude of SMD dropped close to zero after the final attempt (denoted as pink stars). In fact, the mean absolute error (MAE) of the SMDs based on the final attempt had the smallest value (0.036) compared to those based on the first 1 to 5 attempts (0.07 ∼ 0.08).

Note that, because of the relatively small sample sizes, these SMDs are not statistically significant, except for Item 21 (which had a sample size of 46 for the men group and 19 for the non-men group). Item 21 (a task that involved programming skills such as loops, nested branching, and string manipulation) seems to be much harder for the non-men group in all the first 5 attempts, particularly when we

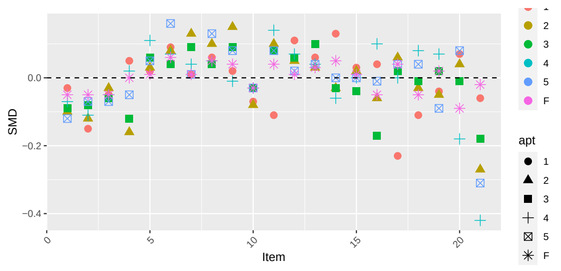


Figure 7: Standardized mean differences (SMDs) between the non-men and men groups after matching on performance scores on Item 1 to Item 21, respectively. The x-axis stands for items (from 1 to 21), and the y-axis on the left stands for SMD (each item has six SMDs based on the six scoring rules). Those six SMDs are reported in different colors and point types (refer to the keys on the right side of the panel). A positive SMD indicates that the item scored by the associated scoring rule may favor the non-men group.

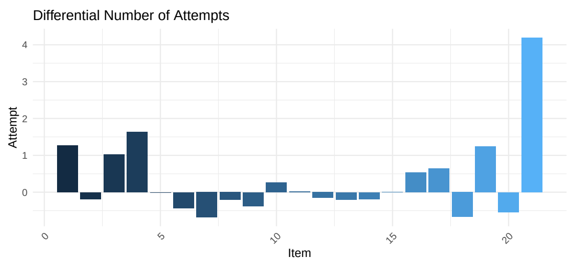


Figure 8: Differential number of item attempts between the non-men and men groups from Item 1 to Item 21. A positive bar indicates that the non-men group made more attempts on the item.

only count the first 4 attempts. However, similar to the other items, the SMD of Item 21 was reduced to almost zero after the final attempt.

Figure 8 shows the differential number of attempts between the two groups. The non-men group attempted more times on Items 1, 3, 4, 16,17, 19, and 21. Particularly on Item 21, the non-men group attempted about 5 more times on average (with a statistical significance) than the men group with comparable programming skills. In other words, the diminished SMD on Item 21 between the two groups might be resulted from significantly more effort and more attempts on this item by the non-men group.

4. Discussion

In this pilot study, we attempted to address an emerging research topic on how to measure learning progress in the adaptive learning and practice platform. We leveraged log data collected from the online learning platform to extract programming practice features and developed a general approach to integrate statistical and psychometric methods and best practice with log data features to measure students' progress in programming practice. The statistical and psychometric methods helped to create dynamic performance scores that captured students' progress during practice and were comparable within individual students who made different attempts and across students who practiced different sets of programming tasks. Such comparability of performance scores not only helps to measure progress in programming practice, but it also helps to evaluate potential task biases between diverse student groups. The current study contributes to methodologies in measuring learning progress and evaluating programming tasks in the dynamic and adaptive online practice setting, which is likely to be applicable to other practice and assessment scenarios.

Our preliminary results may have implications for assessing programming tasks in online practice. First, even in a complex situation where multiple attempts are allowed and different programming tasks are adaptively recommended to students, we can still use features extracted from log files to capture students' programming processes and take advantage of, and develop if necessary, statistical and psychometric methods to assess students' progress and produce comparable performance scores. Most importantly, these practice features and rigorous methods can help identify what tasks may have potential bias and favor one student group over the other. Overall, our preliminary results show that repeated practice with immediate feedback improved all students performance and reduced item biases.

These preliminary findings also suggest assessing students' performance on their final attempt, which gives control back to students and allows them decide how many times they want to practice a task. This may boost students' confidence and improve their performance. As long as students keep trying on a task with feedback, they may eventually pass all test cases, and their effort may help mitigate potential task bias. In addition, our preliminary findings indicate that programming tasks that showed initial bias need more investigation from content perspectives, especially if they are used in contexts that do not allow repeated practice and immediate feedback and that have high stakes.

In terms of recommending tasks for students with different skill levels to practice, performance scores based on the first attempt, or the first few attempts, may have better differentiability than those based on the final attempt, thus they are recommended for targeted classroom instruction, if needed.

One major limitation of the current study was the relatively small sample sizes of the pilot data and limited numbers of programming tasks, which makes it challenging to generalize the preliminary findings to broader CS education scenarios. The team is preparing for a larger scale data collection in the coming year to reexamine the research questions and evaluate whether similar findings remain. In particular, follow-up studies will investigate why some tasks may be biased from content perspectives and aim to provide guidelines for developing fair programming tasks. Further studies will also make use of the fine-grained log data and experiment with machine learning techniques in better understanding students' learning behaviors and programming styles, as well as their association with characteristics of programming tasks.

Finally, we end with implications for CS educators. For CS educators, these preliminary results clearly support formative assessment policies that allow for repeated practice and feedback, to encourage learning and mitigate potential biases in task design that might advantage some groups in CS education. They also suggest that restricting attempts may limit the opportunity of observing students to demonstrate their skills and learn from feedback. Future work should further explore these recommendations, particularly in other learning contexts (e.g., with different forms of feedback, such as auto-graders with hints), other identities (e.g., race, ethnicity, ability), and items (e.g., more complex programming assignments often found in formal education).

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 2055550 and 2100296.

References

[1] J. D. Bransford, A. L. Brown, R. R. Cocking (Eds.), How people learn: Brain, mind, experience, and school, National Academy Press, Washington, DC, 2000.

[2] Q. Hao, D. H. Smith IV, L. Ding, A. Ko, C. Ottaway, J. Wilson, T. Greer, Towards understanding the effective design of automated formative feedback for programming assignments, Computer Science Education 32 (2022) 105–127. URL: https://doi.org/10.1080/08993408.2020.1860408. doi:10.1080/08993408.2020.1860408.

[3] R. Murphy, J. Roschelle, M. Feng, C. A. Mason, Investigating efficacy, moderators and mediators for an online mathematics homework intervention. journal of research on educational effectiveness, Journal of Research on Educational Effectiveness 13 (2020) 235–270. URL: https://doi.org/10.1080/19345747.2019.1710885.

[4] M. Hamilton, A. Luxton-Reilly, N. Augar, et al., Gender equity in computing: International faculty perceptions and current practices, in: ITiCSE '16: Proceedings of the 2016 ITiCSE, Working Group Reports, 2016, pp. 81–102.

[5] A. Oleson, B. Xie, J. Salac, et al., A decade of demographics in computing education research: A critical review of trends in collection, reporting, and use, in: ICER '22: Proceedings of the 2022 ACM Conference on International Computing Education Research, 2022. URL: https://faculty.washington.edu/ajko/papers/OlesonXie2022Demographics.pdf.

[6] R. J. De Ayala, The theory and practice of item response theory, Guilford Press, 2009.

[7] M. J. Davidson, B. Wortzman, A. J. Ko, M. Li, Investigating item bias in a cs1 exam with differential item functioning, in: Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, 2021, pp. 1142–1148.

[8] B. Xie, M. Davidson, A. Ko, Domain experts' interpretations of assessment bias in a scaled, online computer science curriculum, in: Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21), ACM, Virtual Event, USA, 2021.

[9] M. J. Kolen, R. L. Brennan, Test equating, scaling, and linking: Methods and practices, 2nd. ed., Springer-Verlag, New York, NY, 2004.

[10] H. Guo, K. Ercikan, Comparing test-taking behaviors of English language learners (ELLs) to non-ELL students: Use of response time in measurement comparability research, ETS Research Report Series (2021). URL: https://doi.org/10.1002/ets2.12340.

[11] H. Guo, K. Ercikan, Differential rapid responding across language and cultural groups, Educational Research and Evaluation 26 (2021). URL: https://doi.org/10.1080/13803611.2021.1963941.

[12] B. Xie, J. O. Lim, P. K. D. Pham, M. Li, A. J. Ko, Developing novice programmers' self-regulation skills with code replays, in: Proceedings of the 2023 ACM Conference on International Computing Education Research, Volume 1., 2023, pp. 298–313.

[13] K. Yamamoto, H. J. Shin, L. Khorramdel, Multistage adaptive testing design in international large-scale assessments, Educational Measurement: Issues and Practice 37 (2018) 16–27. doi:10.1111/emip.12226.

[14] H. Wainer, R. J. Mislevy, Item response theory, item calibration, and proficiency estimation, in: H. Wainer (Ed.), Computerized adaptive testing: A primer, Erlbaum, 1990, pp. 65–102.

[15] D. Mead, An introduction to multistage testing, Applied Measurement in Education 19 (2006) 185–187. URL: https://doi.org/10.1207/s15324818ame1903_1. doi:10.1207/s15324818ame1903_1.

[16] K. Ercikan, H. Guo, H.-H. Por, Uses of process data in advancing the practice and science of technology-rich assessments, OECD Publishing, 2023. URL: https://www.oecd-ilibrary.org/content/component/7b3123f1-en. doi:https://doi.org/https://doi.org/10.1787/7b3123f1-en.

[17] S. Kim, A comparative study of IRT fixed parameter calibration methods, Journal of Educational Measurement 43 (2006) 353–381. URL: https://doi.org/10.1111/j.1745-3984.2006.00021.x. doi:10.1111/j.1745-3984.2006.00021.x.

[18] H. Guo, M. S. Johnson, D. F. McCaffrey, L. Gu, Practical considerations in item calibration with small samples under multistage test design: A case study, Technical Report RR-24-03, ETS, 2024. URL: https://doi.org/10.1002/ets2.12376.

[19] R. Zwick, A review of ETS differential item functioning assessment procedures: Flagging rules, minimum sample size requirements, and criterion refinement, Research Report No. RR-12-08, 2012. URL: https://doi.org/10.1002/j.2333-8504.2012.tb02290.x.

[20] P. W. Holland, D. T. Thayer, Differential item functioning and the Mantel-Haenszel procedure, in: H. Wainer, H. I. Braun (Eds.), Test validity, Erlbaum, Hillsdale, NJ, 1988, pp. 129–145.

[21] R. P. Chalmers, mirt: A multidimensional item response theory package for the r environment, Journal of Statistical Software 48 (2012) 1–29. URL: https://doi.org/10.18637/jss.v048.i06. doi:10.18637/jss.v048.i06.