Leveraging Psychometric Modeling for Enhancing Programming Skill Assessments

Chen Li*
ETS Research Institute
Educational Testing Service
Princeton, USA
CLi@ETS.ORG
*Corresponding author

Mo Zhang
ETS Research Institute
Educational Testing Service
Princeton, USA
MZhang@ETS.ORG

Xiang Liu
ETS Research Institute
Educational Testing Service
Princeton, USA
XLiu003@ETS.ORG

Hongwen Guo
ETS Research Institute
Educational Testing Service
Princeton, USA
HGuo@ETS.ORG

Amy J Ko
Information School
University of Washington
Seattle, USA
AJKo@UW.EDU

Min Li
College of Education
University of Washington
Seattle, USA
MinLi@UW.EDU

***Abstract*—Computer programming has emerged as a critical 21st century literacy. Programming skills are often assessed with open-ended items that require students to write codes to solve a programming problem. The accuracy of the codes can be evaluated using test cases. These coding items, or tasks, cover different programming concepts, including integer manipulation, looping, and function recursion, each with varying difficulty levels. Despite the prevalence of such assessments, there is a lack of rigorous psychometric analysis on such coding tasks when an unlimited number of attempts are allowed, which is a common practice in assessing programming skills; that is, students can try as many times as possible and debug and revise their codes as needed based on test case results. This study demonstrates an application of psychometric analysis and modeling to a computer programming assessment data set collected from college-level students. In addition to assessing the correctness of the final submitted programs, the analysis considers process data such as time spent on the task and number of attempts. By analyzing both product and process data, we aim to uncover potential associations between different programming concepts and item difficulty level. We propose fitting a cognitive diagnostic model (CDM) with process data as covariates to classify students into different skill mastery patterns. We demonstrate that this modeling approach can provide valuable insights for instructors and researchers in understanding how students learn and master programming concepts.**

***Index Terms*—coding assessment, programming skills, process data, psychometric analysis, cognitive diagnostic model**

I. Introduction

Computer programming has emerged as a critical 21st century literacy. 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. Despite the progress made in understanding the difficulties in learning programming and in interventions for teaching it more effectively, there has been little progress in assessing programming skills. The assessment chapter in a 2019 book [1] that surveyed the entire field of CS education highlights this gap, summarizing a modest body of research on plagiarism, but noting the severe lack of research on designing valid, reliable, and fair measures of programming knowledge. Much of the effort has been put into creating and validating concept inventories or examining logistical issues such as scaling the formative assessment of programming skills (e.g., [2]). Some recent work showed psychometric flaws with these concept inventories [3] and applied modern validity frameworks to assess specific skills [4]. While this progress is meaningful, prior work has overlooked the tensions between assessing the program that a learner produces versus the process that a learner followed to produce that program. An open challenge in assessment in Computer Science education is to understand the relationship between a learner's process and the programs resulting from that process [5].

The objective of this study is to explore the application of psychometric analysis and modeling to a computer programming assessment dataset by leveraging both response process and product data. We used an item and assessment delivery platform that has been developed under the larger research project and allows for data collection of programming product and process information at scale. The platform is capable of capturing all the keyboarding activities and mouse actions executed by a student when writing codes, in addition to the final submitted codes [6]. Using a limited sample size data, we evaluated different ways to evaluate coding performance while incorporating timing and process data. We demonstrate an application of cognitive diagnostic models to uncover potential associations between programming concepts and item difficulty level and as a method to identify the difficult programming skills for students to grasp and classify

students into different skill mastery patterns. We show that this modeling approach can potentially provide invaluable practical insight for instructors and researchers in understanding how students learn and master programming concepts.

II. Methods

A. Cognitive Diagnostic Model Application

Taking into consideration the relative small sample size of the pilot data (N of participants about 180) and the number of programming tasks (i.e., 21) according to [7] and [8], we used generalized DINA (the deterministic inputs, noisy "and" gate) model for dichotomous attributes for our analysis. As suggested in [9] the DINA model is the best choice in a small sample scenario, and its parameters are easy for interpretation.

The DINA model is a popular cognitive diagnostic model, which assumes the test taker must master all required skills associated with that item to answer the item correctly. In our model application, all attributes or skills are assumed to be independent of each other. We estimated the generalized DINA model with monotonicity constraints. Based on students' actual responses, this model turns students' descriptions into a multi-dimensional knowledge point matrix.

As described in [10], in the DINA model with independent attributes, the latent variable $\eta_{ij}(=1/0)$ represents whether respondent j has mastered all attributes required for item i :

$$\eta_{ij} = \prod_{k=1}^{K} \alpha_{jk}^{q_{ik}} \tag{1}$$

where q_{ik} is the element of the Q matrix which is an I×K matrix. q_{ik}=1 if item i requires attribute k and 0 if not. The model allows for slipping and guessing:

$$s_i = \Pr(Y_{ij} = 0 | \eta_{ij} = 1) \tag{2}$$

$$g_i = \Pr(Y_{ij} = 1 | \eta_{ij} = 0) \tag{3}$$

It follows that the probability π_{ij} of a correct response of respondent j to item i is:

$$\pi_{ij} = \Pr(Y_{ij} = 1 | \alpha_j, s_i, g_i) = (1 - s_i)^{\eta_{ij}} g_i^{1-\eta_{ij}} \tag{4}$$

B. Participants and Instrument

Our analysis is based on around 180 students' test records. There were 21 unique programming items in total strategically grouped into a two-stage adaptive test design [6]. The common form in Stage 1 included 9 tasks. There were two forms in Stage 2, one easy form and one hard form. Each form in Stage 2 contained six tasks. Depending on their performance in Stage 1, students were routed to either hard or easy form in Stage 2.

The test data was collected from students across different academic majors and grade levels. On the item delivery interface, the items are presented on the upper-left portion of the screen, students edit and submit their codes on the lower-left area, and check out the running results for each test case on the right side. Besides the final test responses data, processing data was also collected by capturing logs of all keystrokes, program executions, window focus events, copy and paste events, etc. during the coding test.

C. Q Matrix Simplification

Originally, seven programming skills were embedded in our item design, as shown in the Figure 1. They were: integer manipulation, string manipulation, list manipulation, operators, branching, loops, and nested branching. Each item measures a combination of a subset of the seven skills. Based on this design, we got the model's Q matrix, which specifies the attributes measured by each item.

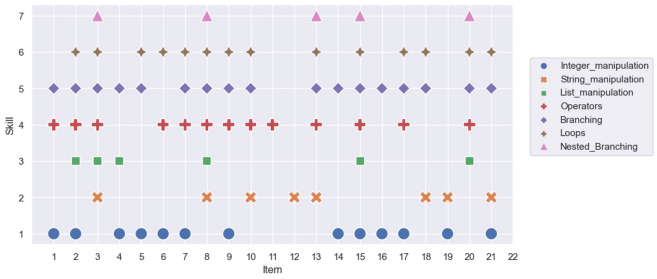

Fig. 1. Programming Skills in Item Design.

Since the sample size was relatively small, in order to get better parameter estimations, we simplified the Q matrix. Specifically, integer manipulation, string manipulation, list manipulation are combined into a single category labeled as "manipulation," and branching and nested branching are grouped together labeled as "branching." As a result, we have a seven-skills frame and a four-skill frame (i.e., manipulation, branching, operators, and loop).

D. Scoring Rules

Traditionally, students' performance was judged based on the accuracy of their final submitted code using test cases and/or a scoring rubric. Recent research has suggested that there are pros and cons from a psychometric perspective to evaluate students' proficiency based on the final outcome [6]. We examined the histogram of the 180 students' response time by items (as showed in Figure 2.), and found that among the students who passed the item in 5 attempts, 78% took less than 5 minutes to complete the code editing. Based on this finding, the following four different scoring rules were used to score the students' coding behavior.

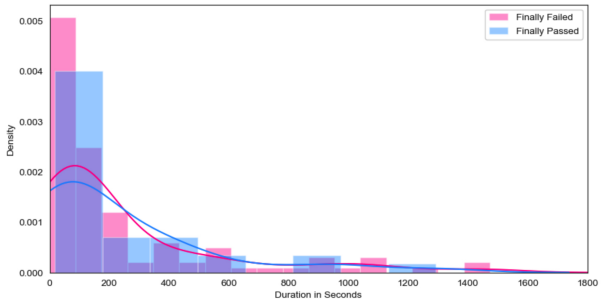

Fig. 2. Histogram of The First Five Attempts Duration.

- **Pass at the first attempt** (Scoring Rule 1): If the code passes all test cases of the item in the first running attempt, then score = 1 on this item; otherwise, 0.
- **Pass within 5 minutes and 5 attempts** (Scoring Rule 2): If the programming code passes all test cases of the item within five minutes and within five running attempts, then score = 1 on this item; otherwise, 0.
- **Pass within 5 attempts** (Scoring Rule 3): If the programming code passes all test cases of the item within five running attempts, then score = 1 on this item; otherwise, 0.
- **Unlimited time and attempts** (Scoring Rule 4): If the programming code passes all test cases of the item, no matter how many attempts the student tried, and no matter how long it took the student, then score = 1 on this item; otherwise, 0.

From the Scoring Rule 1 to Scoring Rule 4, the degree of strictness decreases step by step, the first scoring rule is the most strict and the last one is the most relax with no constraints on the number of attempt or coding time. We computed a total score for each participant by summing all item scores acros. The total score distribution using the four different scoring rules, as can be seen in Figure 3, confirmed the varying degrees of the strictness, with the overall histograms gradually shifted from the left (lower total score, more strict scoring rule) to the right (higher total score, less strict scoring rule).

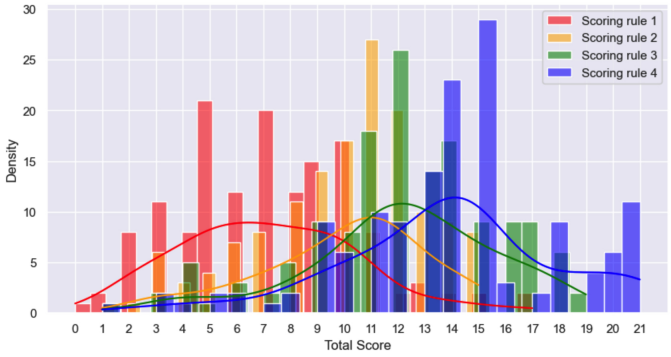

Fig. 3. Total Score Histogram Using The Four Scoring Rules.

III. Results

A. Model Estimation Results

According to the recommendations made in [11] by comparing several software packages, we chose the R software to estimate the DINA model. The analysis used the CDM package for cognitive diagnostic modeling [12] in R. The model fit statistics are given in Figure 4. The models were compared between the four-skill and seven-skill frames, based on AIC, BIC, and mean RMSEA (root mean square error approximation), for each of the scoring rules described above.

From the model fit statistics, we can see that the AIC and BIC values of the seven-skill frame and four-skill frame are very close, while the mean RMSEA values of the four-skill

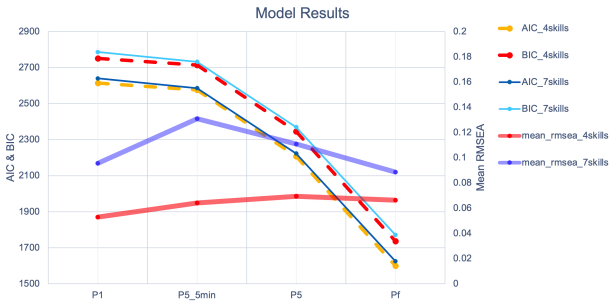

Fig. 4. Model Fit Results

frame (purple line) are lower than the seven-skill frame (orange line). The difference in mean RMSEA between the four- and seven- skill frame was particularly large resulting from the Scoring Rule 2 which includes the response time constraint. Overall, the seven-skill frame RMSEA values between 0.08 and 0.12 are acceptable, but the four-skill frame showed better model fit taking in account of all three evaluation statistics (AIC, BIC, and RMSEA). Therefore, we only report the results based on four-skill frame hereafter.

B. Item Analysis Results

Figure 5 gives the item analysis results using four different scoring rules based on the four-skill frame. The blue bars represent the "guess" parameter estimates by item and the orange bars represent "slip" parameter estimates by item. To interpret the two parameters, a high "guess" estimate means that even students of low proficiency have a high probability to guess the correct answer by chance, a high "slip" estimate means that even students of high proficiency have a high probability to make a careless mistake and fail that item, and vice versa.

The results are rather interesting and generally agreed with our expectations from item design. Overall, items 1 to 13 and item 20 are relatively easy compared to the items 14 to 19. The results also suggested that for items 2, 12, 14 and 21, students needed to exert more effort to figure out the correct coding on these items. And for items 1, 20, and 21, students will improve a lot if they were given longer time. Also there were items (for example, item 17) appeared very difficult at the beginning, but it will finally been resolved if the students tried enough attempts. The "slip" parameters further diminished substantially under the unlimited attempt condition, suggesting that regardless of item difficulty, students are unlikely to make careless mistakes when they were allowed to revise and resubmit their codes as many time as they desired.

C. Demographic Subgroup Analysis Results

As part of the data collection, the students were asked to report their demographic background on a volunteer basis, including their gender identification, ethnicity, home language, prior programming experience, academic major, among others. All questions were provided in an open-ended format. The

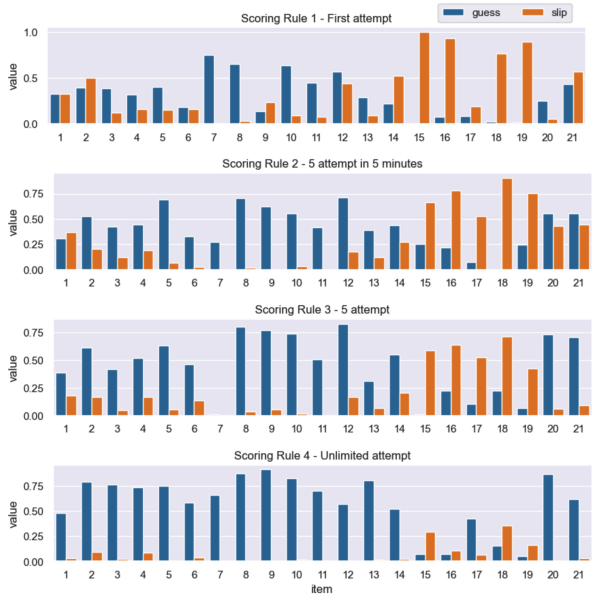

Fig. 5. Item Analysis Based On The Four-skill Frame.

DINA model estimated person parameters which can be used to generate a skill attribute patterns for each individual student.

Here the subgroup analyses are post-hoc analyses based on the estimated person skill profiles from the DINA model. Using the individual parameters estimation we calculated the average estimation scores of each subgroup by self-identified gender and by academic major, respectively. For gender, we compared two subgroups: the "men" group which is the dominant subgroup in CS education and the "non-men" group. For academic major, four subgroups were compared: computer science, engineering, math, and others. The subgroup comparison results are given in the following spider graphs. For the purpose of demonstration, we only show the comparisons between the most strict Scoring Rule 1 (passing all test cases at first attempt) and the least strict Scoring Rule 4 (passing all test cases with unlimited time and attempt).

In Figure 6, the solid lines represented the subgroups' average ability evaluated by first attempt scores on the four skills, and the dashed lines represent the subgroups' average ability evaluated by the final scores. The blue color represents subgroup of men and the red color represents non-men. At the beginning, using Scoring Rule 1, the men group's appeared to outperform the non-men group, especially on the "loops" skill. However with more attempts and unlimited time on task, the results based on Scoring Rule 4 shows that the non-men group appears to have caught up and there is much less obvious difference between the men and non-men groups on any of the four skills. Furthermore, the gaps between the solid lines and dashed lines suggest that allowing for unlimited time on task and attempts appeared to have the biggest impacts on the estimations on the loops skill.

Figure 7 shows the performance comparisons between students in different academic majors. The solid lines represent

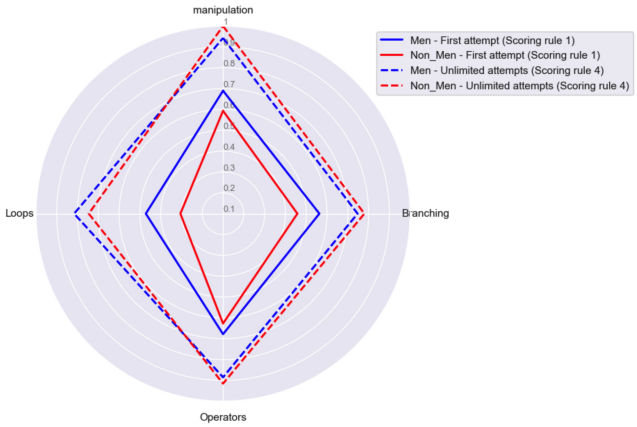

Fig. 6. Skill Evaluation By Gender Based on Four Skills.

the mean estimated ability for a subgroup evaluated by Scoring Rule 1 and the dashed lines represent the mean estimated ability evaluated by Scoring Rule 4. We find that, for the loops, manipulation and branching skills, the computer science students appear to perform the best when evaluated by Scoring Rule 1, the "Math" and "Other" major students in the middle, and the "Engineering" students the worst at the beginning. There is no obvious difference on the operators skill among the four major groups. The patterns shifted a bit in the final attempt scores resulting from Scoring Rule 4. In the end, the "Engineering" major group demonstrated better loops and branching skills compared to the other three major groups. Although the other three major groups also demonstrated better skills overall in the final attempt, the changes were generally not as substantial as the "Engineering" major group.

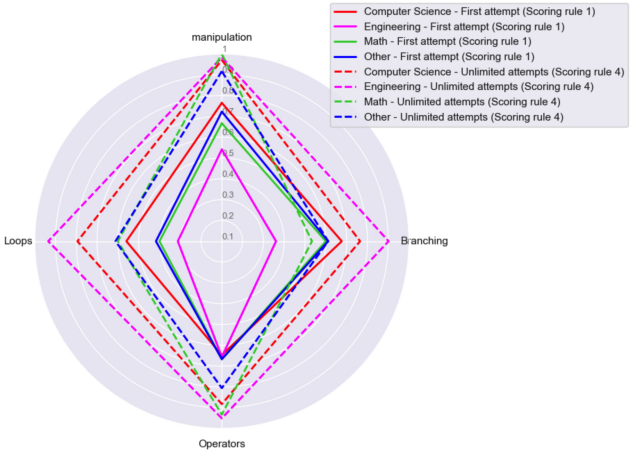

Fig. 7. Skill Evaluation By Major Based On Four Skills.

IV. Summary and Discussion

Learning computer programming has become essential not only for career development but also as a means of comprehending the increasing influence of computing technologies on society. Despite the strides made in programming education,

there is still a noticeable gap in developing reliable methodologies to assess programming skills. This study seeks to address this shortfall by an empirical demonstration of applying rigorous psychometric models and analyses to a computer programming assessment dataset, utilizing data on both the end product and the programming process implemented by individuals. A platform was developed to capture the full breadth of student interaction with programming tasks, including keyboard and mouse activities, enabling a comprehensive collection of data that captures both the final code and the coding process. A cognitive diagnostic model, specifically the generalized DINA model, was applied to this data to explore the relationship between programming concepts and task difficulty, providing new insights into how students learn and master programming skills.

The analysis simplified the original seven-skill model into a four-skill model by grouping related skills to achieve better model fit due to the small sample size. Various scoring rules were examined, establishing a gradient of strictness that evaluated how constraints on time and number of attempts affected students' performance. Results indicated the potential for improved performance when allowances were available for additional time and attempts. Furthermore, demographic subgroup analyses revealed nuanced differences in performance based on gender and academic major, with differing patterns observed in students' mastery of skills such as loops and branching. Using these analyses, we highlight the value of using psychometric models in coding assessments to uncover deeper insights into student learning processes and skill acquisition in programming.

One of the primary limitations of this study is the relatively small sample size of approximately 180 participants, which may not be representative of the broader population of programming learners. This limitation could affect the generalizability of the findings, as a more diverse and larger sample size might yield different insights into the complexities of programming skill acquisition. As the small sample size could also affect the reliability of the CDM results, in the future, we can examine its impact using the standard errors of the parameter estimates and estimating the classification uncertainty as suggested in [13]. Additionally, the simplification from a seven-skill to a four-skill model, driven by the need for more reliable parameter estimations, might have overlooked some nuances in the skill sets required for programming, potentially oversimplifying the relationships between different skills. The reliance on digital logs to capture the coding process also raises concerns regarding the completeness and accuracy of data, as these logs may not precisely reflect a learner's cognitive processes or problem-solving strategies. Moreover, the study utilized a limited number of programming tasks, which might not encompass the full breadth of programming skills and concepts typically involved in a comprehensive computer science curriculum. Lastly, the conclusions drawn from demographic subgroup analyses might be constrained by the self-reported nature of demographic data, which can be prone to inaccuracies and inconsistencies.

The next phase of this research will involve the collection of additional data to enhance the model-based inferences and to gather detailed evidence on the reasons behind errors in each coding attempt. This will allow for expanding detailed item-level analysis such as in [6] and a more thorough analysis of the coding process data, potentially uncovering patterns that inform better teaching methods. Additionally, we are exploring the feasibility of automatically generating personalized feedback for students, tailored according to their performance and specific challenges identified through the data. Furthermore, we will investigate the relationship between task variation and coding processes, particularly examining these dynamics from a fairness perspective. This entails ensuring that all students have the same opportunity to demonstrate their abilities, leading to more inclusive and effective educational practices.

ACKNOWLEDGMENTS

This research is supported by the National Science Foundation Grant No. 2055550 and 2100296.

REFERENCES

[1] S. A. Fincher & A. V. Robins (Eds.). (2019) "The Cambridge handbook of computing education research." Cambridge University Press.

[2] B. Xie, D. Loksa, G. L. Nelson,M. J. Davidson, D. Dongsheng, H. Kwik, A. H. Tan, L. Hwa, M. Li, & A. J. Ko. (2019) "A theory of instruction for introductory programming skills." Computer Science Education, 26(2–3), 205–253.

[3] Xie, B., Davidson, M. J., M. Li, & A. J. Ko. (2019) "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 (pp. 699–705). ACM.

[4] G. L. Nelson, A. Hu, B. Xie & A. J. Ko. (2019) "Towards validity for a formative assessment for language-specific program tracing skills." In Koli Calling '19: Proceedings of the 19th Koli Calling International Conference on Computing Education Research (pp. 1-10). ACM.

[5] E. Thompson, A. Luxton-Reilly, J. L. Whalley, M. Hu & P. Robbins. (2008) "Bloom's taxonomy for CS assessment." In Proceedings of the Tenth Conference on Australasian Computing Education-Volume 78. 155–161.

[6] H. Guo, M. Zhang, A. J. Ko., M. Li, B. Zhou, J. Lim, P. Pham & C. Li. (2024) "Measuring Students' Programming Skill via Online Practice." In Proceedings of the 8th Educational Data Mining in Computer Science Education (CSEDM) Workshop. Atlanta, GA.

[7] G. Uyumaz & O. Çokluk-BÖKEOĞLU. (2017) "Effect of Q-matrix misspecification on parameter estimation in differing sample sizes and test length for DINA." EBAD - JESR, 7(1), 91–108.

[8] T. Basokcu. (2014) "Classification Accuracy Effects of Q-Matrix Validation and Sample Size in DINA and G-DINA Models." Journal of Education and Practice; Vol 5, No 6 2014; 220-230.

[9] M. A. Sorrel, S. Escudero,P. Nájera, R. S. Kreitchmann & R. Vázquez-Lira. (2023) "Exploring Approaches for Estimating Parameters in Cognitive Diagnosis Models with Small Sample Sizes." Psych, 5(2), 336-349.

[10] SY Lee. (2016) "DINA model with independent attributes." Technical report, University of California at Berkeley. https://mc-stan.org/users/documentation/case-studies/dina_independent.html.

[11] S. Sen & R. Terzi. (2020) "A Comparison of Software Packages Available for DINA Model Estimation." Applied Psychological Measurement, 44(2), 150–164.

[12] A. C. George, A. Robitzsch, T. Kiefer, J. Groß & A.Ünlü. (2016) "The R Package CDM for Cognitive Diagnosis Models." Journal of Statistical Software, 74(2), 1–24.

[13] M. S. Johnson, & S. Sinharay. (2018) "Measures of agreement to assess attribute-level classification accuracy and consistency for cognitive diagnostic assessments." Journal of Educational Measurement, 55(4), 635-664.