Education and Information Technologies
https://doi.org/10.1007/s10639-024-13039-6

Journal Article

Profiling the Skill Mastery of Introductory Programming Students: A Cognitive Diagnostic Modeling Approach

Manuel B. Garcia [a,b,c] *

[a] College of Education, University of the Philippines Diliman, Quezon City, Philippines
[b] Educational Innovation and Technology Hub, FEU Institute of Technology, Manila, Philippines
[c] College of Education, Korea University, Seoul, South Korea

*** Correspondence:**
Manuel B. Garcia, University of the Philippines Diliman and FEU Institute of Technology.
mbgarcia@feutech.edu.ph

How to cite this article:
Garcia, M. B. (2024). Profiling the Skill Mastery of Introductory Programming Students: A Cognitive Diagnostic Modeling Approach. *Education and Information Technologies*. https://doi.org/10.1007/s10639-024-13039-6.

Article History:
Received: 27 Feb 2024
Revised: 22 Mar 2024
Accepted: 3 Sep 2024
Published: 8 Oct 2024

Abstract:

The global shortage of skilled programmers remains a persistent challenge. High dropout rates in introductory programming courses pose a significant obstacle to graduation. Previous studies highlighted learning difficulties in programming students, but their specific weaknesses remained unclear. This gap exists due to the predominant focus on the overall academic performance evaluation. To address this gap, this study employed cognitive diagnostic modeling (CDM) to profile the skill mastery of programming students. An empirical analysis was conducted to select the most appropriate model for the data, and the linear logistic model (LLM) was determined to be the best fit. Final examination results from 308 information technology (IT) and 279 computer science (CS) students were analyzed using the LLM. Unfortunately, findings revealed that programming students exhibited proficiency primarily in code tracing and language proficiency but displayed deficits in theoretical understanding, logical reasoning, and algorithmic thinking. From a practical standpoint, this deficiency in fundamental skills sheds light on the factors contributing to academic failures and potentially eventual dropout in programming education. When comparing the student population by academic program, CS students demonstrated superior mastery compared to their IT counterparts, although both groups exhibited a lack of mastery in code tracing. These deviations underscore the pressing need for tailored educational strategies that address the unique strengths and weaknesses of each student group. Overall, this study offers valuable insights into programming education literature and contributes to the expanding application of CDM in educational research.

Keywords:

Cognitive Diagnostic Modeling, Programming Education, Information Technology, Computer Science, Educational Assessment

INTRODUCTION

Computer programming is an essential skill in modern society. According to the Asian Development Bank (2022), programming skills are becoming increasingly essential in all job sectors worldwide. This trend is unsurprising, considering the ongoing digitalization and the growing complexity of technological solutions in various industries. Many non-coding job positions also now demand proficiency in computer programming. The International Labour Organization (2021) posited that coding skills today are required not only of programmers but also of scientists, engineers, designers, and artists. This demand underscores the necessity of integrating these skills into educational curricula. Governments across the globe have responded by adapting their educational systems. This adaptation involves the systematic incorporation of programming courses across different levels of education (Ou et al., 2023), in addition to offering them within information technology (IT), computer science (CS), and other computing programs. For instance, Macrides et al. (2022) noted a significant trend in early childhood education towards teaching coding. Their systematic review highlighted the adoption of screen-based visual programming and robotics for this purpose. This trend continues into middle school, where Lira et al. (2022) identified programming camps as a significant supplemental educational tool. It also extends to higher education, where Agbo et al. (2019) observed the increasing integration of computational thinking in teaching problem-solving skills and programming education. By incorporating these skills into the curricula, countries are preparing future generations for the challenges and opportunities of an increasingly digital world.

Despite this global shift in education, the world continues to face a shortage of skilled programmers. In a report covering four major countries (i.e., Canada, China, Germany, and Singapore), the International Labour Organization (2020) has identified substantial deficits in the number of software developers and programmers. Similar widespread shortages have been observed by the European Labour Authority (2023) in the EU27, Norway, and Switzerland. This global issue is multifaceted, but a critical component centers around the challenges within education systems. Programming education often presents significant learning difficulties for students. Garcia (2021) asserted that these challenges are influenced by individual differences (e.g., inherent aptitudes and learning styles) as well as cognitive (e.g., problem-solving skills and logical reasoning abilities) and non-cognitive (e.g., motivation and attitude) factors. The effectiveness of programming education is also highly dependent on the quality of teaching and curriculum design. However, Ou et al. (2023) observed that the current quality of programming education is lacking, and there is a need for enhanced curriculum development in schools. This situation underscores the importance of rigorous assessment in programming education. Such assessments are vital in identifying areas where students face challenges. They also guide the refinement of teaching methods and curriculum to ensure they are aligned with the evolving needs of learners. However, most assessments are focused on measuring overall performance rather than diagnosing specific cognitive strengths and weaknesses (Garcia & Revano, 2021). This general approach tends to overlook critical insights into individual learning attributes, which is crucial for addressing the unique challenges each student faces.

This gap in traditional assessment methods highlights the need for an approach that can identify the cognitive abilities of programming students. While traditional assessments measure

overall performance, they often fail to diagnose specific cognitive strengths and weaknesses. This limitation makes it difficult to provide targeted support. To address this issue, Cognitive Diagnostic Modeling (CDM) emerges as a fitting solution. CDM is a sophisticated analytical approach that focuses on understanding and diagnosing the specific cognitive skills and knowledge structures individuals possess. By providing detailed and multidimensional diagnostic feedback, CDM can identify examinees' strengths and weaknesses across a spectrum of attributes. This technique can be particularly beneficial in programming education, where understanding the intricacies of a student's cognitive abilities can lead to more effective instructional strategies. Applying a CDM approach can inform curriculum development and optimize learning outcomes by aligning instructional methods with the diverse cognitive needs of students. Therefore, this study seeks to address the following research questions (RQ):

RQ1. Which cognitive diagnosis model most adequately fits the empirical data?
RQ2. What are the attributes mastered by students at the grade and individual levels?
RQ3. How do these mastery profiles vary between CS and IT students?

BACKGROUND OF THE STUDY

Cognitive Diagnosis Modeling

CDM is an advanced analytical approach designed to understand and diagnose the specific cognitive skills and knowledge structures that individuals possess. Its foundational roots date back to the development of the rule space method (Tatsuoka, 1983). Over the years, CDM has evolved to incorporate various models and techniques aimed at providing detailed and multidimensional diagnostic feedback (Liu et al., 2023). Unlike traditional psychometric frameworks that are more descriptive, such as item response theory (IRT) and classical test theory (CTT), CDM offers a diagnostic framework that classifies examinees' strengths and weaknesses across a spectrum of attributes (de la Torre & Minchen, 2014). In this context, an attribute is described as essential knowledge and cognitive abilities crucial for solving specific problems or tasks. For example, a CDM could indicate whether students learning programming have mastered specific attributes essential to coding, such as understanding basic syntax, applying control structures such as loops and conditionals, or efficiently debugging code (Garcia, Enriquez, et al., 2022). This level of granularity provides more detailed evidence than other psychometric models, making CDM particularly useful for guiding teaching and learning decisions in the classroom (Effatpanah et al., 2019; Paulsen & Valdivia, 2022). Given its capabilities as a psychometric model, CDM is frequently used as the analytical framework in cognitive diagnostic assessment as it offers a more comprehensive evaluation of students' learning processes (Li et al., 2021).

The application of CDM has shown significant success in various educational contexts, including reading (Jang et al., 2015), listening (Meng et al., 2023), writing (Effatpanah et al., 2019), mathematics (Chandía et al., 2023), and accounting (Helm et al., 2022). These studies have demonstrated the effectiveness of CDM in providing detailed insights into specific skill sets and cognitive abilities of students. However, despite the empirical evidence demonstrating the benefits of CDM in other educational domains, it has not yet been adopted in programming education. A review of the literature reveals a significant gap, as no studies have specifically

employed CDM as an assessment approach in programming education. The closest prior work involves the assessment of computational thinking, which has a broader focus on problem-solving and algorithmic reasoning rather than programming-specific skills (Li & Traynor, 2022). Unfortunately, traditional assessments used in programming education (e.g., Qayyum et al., 2018; Schnieder & Williams, 2022), while useful in evaluating the general understanding and competence of learners, often fail to diagnose specific cognitive strengths and weaknesses. In contrast, CDM offers a unique opportunity to identify the nuances of students' cognitive abilities. By diagnosing specific areas where students may struggle or excel, CDM can provide educators with the insights needed to tailor instruction more effectively. Given the limitations of traditional assessment methods in programming, there is a compelling need to explore and integrate CDM into this field to enhance both learning outcomes and instructional practices.

Foundational Models in Cognitive Diagnosis

An important consideration in applying a CDM is selecting an appropriate model (Wu et al., 2024). CDM encompasses various types of models, each with unique features and applications. Saturated models, such as the G-DINA (*generalized deterministic inputs, noisy "and" gate*) model (de la Torre, 2011), are the most comprehensive, as they allow for the estimation of all possible interactions among attributes. These models are highly flexible and can capture complex relationships, but they require a large amount of data and can be computationally intensive. Conversely, constrained models simplify the structure by assuming that certain interactions are negligible, thus reducing the number of parameters to be estimated. Examples of constrained models include the DINA (*deterministic inputs, noisy "and" gate*) model (Junker & Sijtsma, 2001), the DINO (*deterministic input, noisy "or" gate*) model (Templin & Henson, 2006), additive CDM (ACDM; de la Torre, 2011), linear logistic model (LLM; Maris, 1999), reduced reparameterized unified model (RRUM; Hartz, 2002), and more. These models are easier to manage and interpret but may not capture all the nuances of the data. When there are uncertain relationships among attributes, Ma and de la Torre (2020b) noted that the higher-order GDINA model with Rasch, 1-Parameter Logistic (1PL), and 2-parameter Logistic (2PL) joint attribute distributions can be considered to select appropriate models for empirical studies.

In some cases, a mixed model approach is used as a supplement to standard CDM analysis, where different models are applied to individual items within the same assessment. This technique allows for a tailored analysis that can better fit the varying complexities of various questions in an instrument. For instance, simpler items might be analyzed with reduced models, while more complex items might require saturated models to fully capture the cognitive processes involved. In a practical application, Ravand and Robitzsch (2018) applied this method in a reading comprehension context and found that a mixed model provided a better fit than the G-DINA model. Given the abundance of viable models, de la Torre and Lee (2013) argued that objectively choosing the most appropriate model is crucial rather than relying on personal preference or a predetermined model. As a guiding approach, the parsimony principle suggests selecting the simplest model when faced with multiple statistically equivalent models. However, model selection should also be based on how well the model assumptions correspond to the theoretical basis used to construct a given test (Li et al., 2015). de la Torre and Lee (2013) noted that the Wald Test, a statistical test for parameter significance, can be used to compare models

under the G-DINA framework. Using this test allows the selection of the model that best fits the specific context of the assessment. The choice of model impacts the accuracy and utility of the diagnostic information obtained, making it essential to consider the characteristics of the items and the attributes being measured (Effatpanah et al., 2019; Helm et al., 2022). This careful selection ensures that the CDM approach is effectively tailored to provide fine-grained diagnostic information and the most meaningful insights into students' cognitive abilities and learning needs.

METHODS

Study Setting and Participants

The research was conducted at one of the leading institutes of technology in the Philippines. This university hosts a College of Computer Studies and Multimedia Arts (CCSMA), which offers IT and CS undergraduate programs. A fundamental component shared between these programs is a series of introductory and advanced computer programming courses. One of the programming courses that plays a significant role in the curriculum of both programs is Computer Programming 1, which comprises lecture (CCS0003) and laboratory (CCS0003L) components. The primary objective of this introductory programming course is to teach first-year computing students the foundational skills in computational logic and design. The course covers traditional problem-solving techniques (e.g., flowcharting and pseudo-coding) and basic programming concepts covering input/output operations, conditional and repetitive control structures, and arrays. Garcia (2021) utilized the same course in conducting experimental research on evaluating cooperative learning pedagogy in computer programming. The selection of this course for the study is strategic, as it represents a shared educational experience for IT and CS students. The course maintains uniformity in its syllabus, teaching materials, and online modules, ensuring instruction consistency across different faculty members and between the two programs. It also guarantees that all computing students are assessed under similar conditions, making the evaluation of their skills and knowledge fair and unbiased. By maintaining uniformity in course content and delivery, the research design effectively controls for extraneous variables that might otherwise influence the outcome of the study (Garcia, 2023).

Research Instrument and Data Collection

This study utilized a comprehensive 100-item multiple-choice final examination from the CCS0003 course. Administered during the first trimester of the academic year 2023-2024, all IT and CS students enrolled in the course took this departmental examination for an hour. The instrument development was spearheaded by the faculty-in-charge, with subsequent validation by a team of co-faculty members who also teach the course. This collaborative approach in the instrument's development and validation ensured its academic rigor and alignment with the course's educational objectives. It is important to note that, although arguably better approaches to assess students exist (e.g., practical coding assessments), the number of items and the multiple-choice format are departmental requirements. Despite the multiple-choice format, several questions presented students with scenarios involving machine problems requiring them to interpret and analyze provided code snippets. Successfully responding to these questions necessitates a comprehensive understanding of the underlying algorithms. Additionally, this

instrument was created simply as a final course assessment and not specifically for CDM analysis. Lee et al. (2012) argued that very few assessments are designed based on a cognitive diagnosis framework. More commonly, CDM is applied retrospectively to assessments initially developed with a unidimensional item response theory framework (i.e., retrofitting).

Nonetheless, the primary reason for selecting the final examination is the availability of detailed data (i.e., student responses on an item-by-item basis and the correctness of these responses). The dataset was readily accessible through *ZipGrade* – a mobile optical scanner application for grading multiple-choice assessments. The administrative office of the CCSMA was formally requested to provide a copy of the examination and the results. In response to the request, and with consideration for ethical research practices, they provided randomly selected data from various IT (n = 308) and CS (n = 269) classes. Upon receipt of the data, the first step was anonymizing it to ensure student confidentiality. This anonymization process involved removing all personally identifiable information, such as names, identification numbers, and any other markers traceable to individual students. This step was crucial for protecting student privacy and upholding the integrity of our research. More importantly, this approach strictly complied with data protection regulations and institutional ethical guidelines.

Table 1. Attributes Required for the CCS0003 Final Examination

Attributes	Definition	References
Theoretical Understanding	Deep understanding of the principles and theories that form the foundation of programming.	(Garcia & Revano, 2021; Hota et al., 2023; Thuné & Eckerdal, 2019)
Language Proficiency	Mastery in using programming languages effectively to solve problems and create applications.	(Garcia, Enriquez, et al., 2022; Guo, 2018; Xie et al., 2019; Zhang et al., 2023)
Logical Reasoning	Ability to apply coherent and rational thinking to solve problems and make decisions in programming.	(Barlow-Jones & van der Westhuizen, 2017; Djurdjevic-Pahl et al., 2017)
Algorithmic Thinking	Skill in designing, understanding, and implementing instructions to solve specific problems efficiently.	(Angeli, 2022; Kiss & Arki, 2017; Lamagna, 2015; Tsukamoto et al., 2017)
Code Tracing	Competence in following a program's execution flow and comprehending the behavior of the code.	(Kumar, 2015; Russell, 2022; Stankov et al., 2023; Zhang et al., 2023)

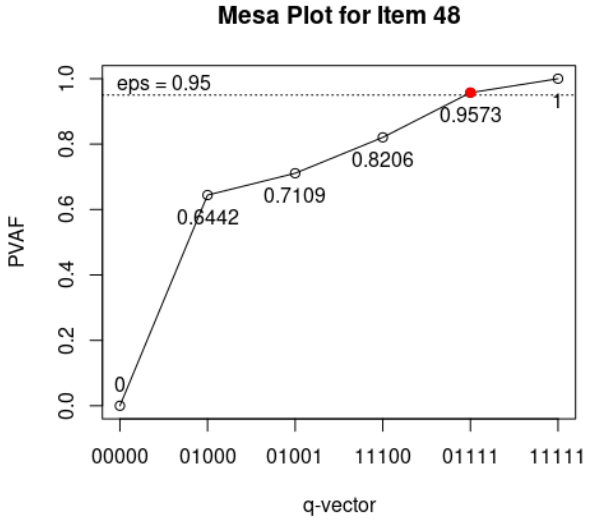
Q-Matrix

Upon obtaining a copy of the examination, domain expertise was utilized, enriched by insights from relevant literature (e.g., Xie et al., 2019), to identify the essential attributes that computer programming students must possess. At this point, the primary goal was to compile a comprehensive list of these attributes. Then, the examination was reviewed item-by-item to check for missing attributes or to confirm that all identified attributes were covered. Indeed, specific attributes (e.g., debugging and code documentation) were not included in the study due to the absence of corresponding questions in the examination. The initial list of attributes was then presented to the faculty-in-charge who developed the examination. Following a consultation, a consensus was reached that the attributes measured by this examination include theoretical understanding, language proficiency, logical reasoning, algorithmic thinking, and code tracing (see Table 1). Subsequently, a Q-Matrix was developed to illustrate the relationship between

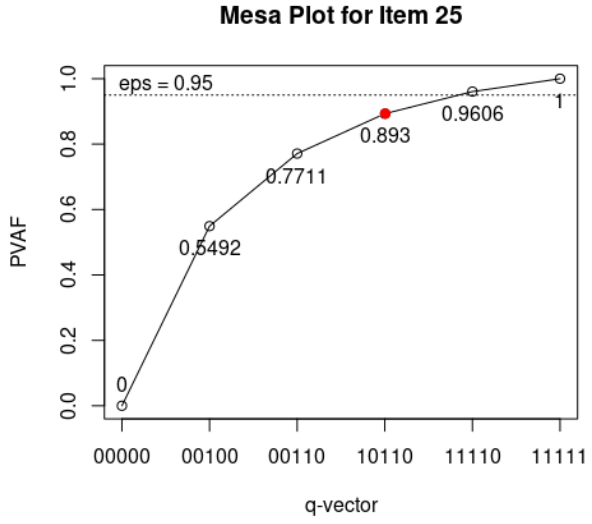
examination items and the identified attributes. Mapping the test items onto an item-by-skill table is a critical first step in CDM (Tatsuoka, 1983). The Q-Matrix underwent validation by the faculty team responsible for the creation and validation of the CCS0003 examination. In cases of disagreement, conflicting viewpoints were discussed and resolved through collaborative decision-making to ensure a unified and accurate representation in the matrix. Several revisions were made based on their feedback, and the revised version served as our initial Q-Matrix.

Data Analysis

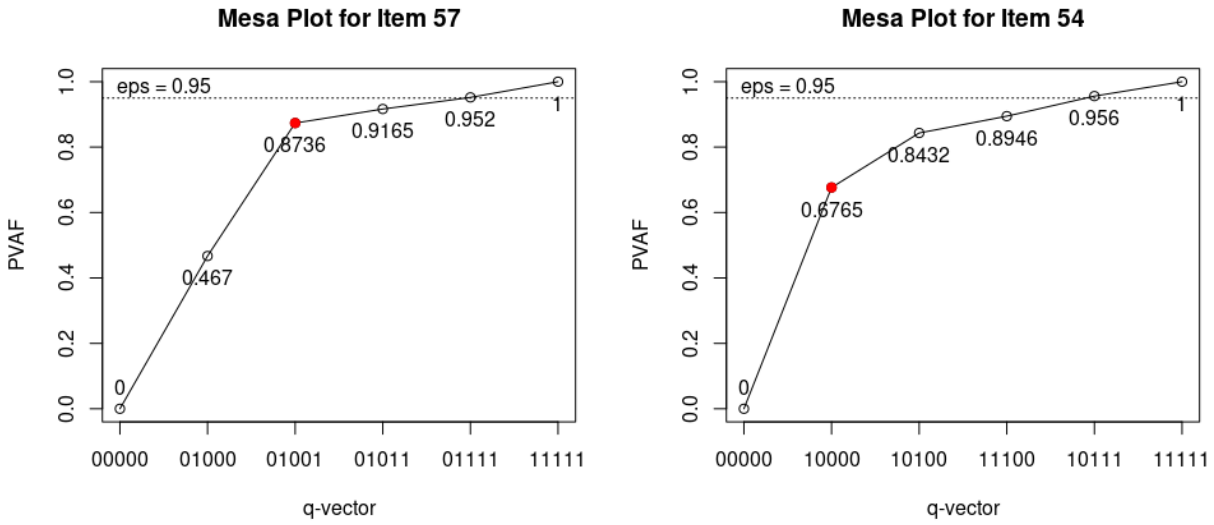
Following the development of the initial Q-matrix, data analysis was conducted using the R programming language, employing the *GDINA* framework (Ma & de la Torre, 2020b) as well as the *tidyr*, *ggplot2*, and *fmsb* packages. The data analysis was initiated with an empirical validation of the item-by-skill table. Ma and de la Torre (2020a) have observed that Q-matrices developed by domain experts often tend to be subjective, which is why it is critical to validate them empirically to avoid erroneous attribute estimation. The results provided by the G-DINA model were consulted using the Proportion of Variance Accounted For (PVAF) with a cutoff greater than 0.95 (de la Torre & Chiu, 2016). Additionally, the mesa plots (refer to Figure 1) of items flagged for revision were manually checked for further analysis. Revisions were made only when they were logically consistent with the item and the skills required for its correct response. This validation process led to the finalization of the Q-Matrix, with the results indicating that 86 out of 100 q-vectors were retained (e.g., Item 48; Figure 1a). Regarding the 14 items with suggested q-vector modifications: six items had one suggested change each (e.g., Item 25: from 10110 to 11110; Figure 1b), six items had two suggested changes each (e.g., Item 57: from 01001 to 01111; Figure 1c), and one item had three suggested changes (e.g., Item 54: from 10000 to 10111; Figure 1d. The final and validated Q-matrix can be found in Appendix A.



(a) Retained Attributes



(b) One Attribute Revised



(c) Two Attributes Revised

(d) Three Attributes Revised

Figure 1. Mesa Plots of Sample Items Showcasing Varying Levels of Attribute Modifications

Afterward, the analysis progressed by fitting the G-DINA model while imposing the monotonicity constraints on the dataset. This saturated model provided a baseline for our analysis. Subsequently, various models were explored, including the DINA model, the DINO model, ACDM, LLM, and RRUM. Given the diversity of cognitive processes involved, it may be more beneficial to avoid forcing a single model onto the entire dataset. Recognizing the complexity of cognitive processes and preventing the imposition of a single model on the entire test, an item-level model fit analysis was conducted. This approach allowed us to consider how each model applied to individual test items rather than the entire test. To select the most appropriate model at the item level, this study followed the process outlined by Ma et al. (2016). First, the Wald statistic for all models for every item was calculated. de la Torre and Lee (2013) recommended the use of the Wald test as an objective means of determining the most appropriate models. In this approach, the *null* hypothesis posits that the reduced model fits the item as well as the saturated model. If the *null* hypothesis is rejected ($p < .05$), the reduced model is dismissed. If more than one reduced model is retained and DINA or DINO is among them, the one with the most significant *p*-value is selected. The outcome of this analysis was a mixed model (subsequently referred to as MIXED), which combined different models at an item level. In addition to these models, we incorporated higher-order G-DINA models such as Rasch, 1PL, and 2PL into our analysis. Several studies have demonstrated the potential of using higher-order models in examining the skill profiles of students (e.g., Zhang et al., 2022).

All these models were included in the relative fit analysis, where the performance of saturated, reduced, mixed, and higher-order models was compared using the *anova()* function in the G-DINA framework. This comparative analysis was pivotal in selecting the most appropriate model. Models that were not rejected during this analysis were further examined, and the one with the lowest Akaike Information Criterion (AIC; Akaike, 1974) and Bayesian Information Criterion (BIC; Schwarz, 1978) was selected as the most suitable model. Both AIC and BIC are

relative fit indices used for selecting between non-nested models, with lower values indicating a better fit. These indices provide a balance between model complexity and goodness of fit, helping avoid overfitting while ensuring accuracy (Chen et al., 2013). After determining the best model, the focus shifted to evaluating the prevalence of various cognitive attributes among programming students. This step was essential for understanding the commonalities and variations in cognitive skills within the student population. Finally, a multi-group comparison was conducted to further assess the cognitive attributes of CS and IT students. This analysis aimed to identify significant differences in abilities and skills between the two student cohorts, which can provide valuable insights into the specific educational needs of each discipline.

RESULTS

RQ1: Which cognitive diagnosis model most adequately fits the collected data?

Table 2 presents the results of the item-level model fit analysis. The model selection process at the item level was guided by statistical criteria to ensure that only the most fitting models were retained for analysis. For instance, both DINA and DINO were retained for Item 13, and DINO was chosen because it had a higher p-value than DINA. In cases where DINA and DINO have the same value (e.g., item 67), either of the two can be selected. Conversely, if neither DINA nor DINO are among the retained models, the reduced model (be it ACDM, LLM, or RRUM) with the most significant p-value is selected. For example, both DINA and DINO were dismissed for Item 48, and LLM was chosen as it had the highest p-value. It should be noted that when several reduced models have p-values greater than .05, DINA or DINO are preferred over other models due to their statistical simplicity (Rupp & Templin, 2008). Finally, items measuring a single attribute that was not included in the table used G-DINA by default.

Table 2. p-values of Each Reduced Model Obtained Using the Wald Test

Items	DINA	DINO	ACDM	LLM	RRUM	Selected Model
Item 13	0.9747	0.9791	0.9882	0.9691	0.9952	DINO
Item 19	1.0000	0.9956	0.9732	0.9578	0.9772	DINA
Item 20	0.9971	0.9988	0.9545	0.9796	0.9590	DINO
Item 22	0.9909	0.9956	0.9907	0.984	0.9803	DINO
Item 23	0.9977	0.9964	0.9771	0.9964	0.9747	DINA
Item 32	0.9997	1.0000	0.9990	1.0000	0.9998	DINO
Item 40	1.0000	0.9995	1.0000	1.0000	1.0000	DINA
Item 41	0.9892	0.9579	0.9954	0.9936	0.9972	DINA
Item 48	0.0000	0.0000	0.9997	1.0000	0.0000	LLM
Item 51	0.0003	0.0000	0.0000	1.0000	1.0000	LLM
Item 52	0.0283	0.4828	1.0000	1.0000	0.0220	DINO
Item 53	0.0000	0.8404	1.0000	1.0000	0.9960	DINO

Item 57	0.0000	0.9531	0.9864	0.999	0.0000	DINO
Item 58	0.0000	0.0000	0.9981	0.9998	0.4145	LLM
Item 59	0.0000	0.9337	0.9992	1.0000	0.0057	DINO
Item 61	0.0000	0.0000	1.0000	1.0000	0.0351	LLM
Item 62	0.0000	0.0000	0.9996	1.0000	0.5883	LLM
Item 63	0.1634	0.1247	0.001	1.0000	1.0000	DINA
Item 64	0.0004	0.0004	0.0000	1.0000	1.0000	LLM
Item 65	0.0000	0.0000	0.0000	1.0000	0.0000	LLM
Item 67	0.8562	0.8562	0.049	1.0000	1.0000	DINO
Item 68	0.0000	0.9988	0.4478	1.0000	0.0000	DINO
Item 69	1.0000	1.0000	1.0000	1.0000	1.0000	DINO
Item 72	0.0266	0.0000	1.0000	1.0000	1.0000	LLM
Item 73	1.0000	0.6735	0.9243	0.0000	0.0000	DINA
Item 75	0.9983	0.9836	0.0000	1.0000	1.0000	DINA
Item 77	0.9356	0.0000	0.0000	0.0001	0.0000	DINA
Item 78	0.0015	0.0000	1.0000	1.0000	1.0000	LLM
Item 79	0.0000	0.0000	0.9946	0.9997	0.8927	LLM
Item 80	0.9986	1.0000	0.9998	1.0000	1.0000	DINO
Item 81	1.0000	1.0000	0.9998	0.0000	0.0000	DINO
Item 83	0.2527	0.0948	1.0000	1.0000	1.0000	DINA
Item 84	0.9985	0.9980	0.2712	1.0000	1.0000	DINA
Item 85	0.9612	0.9611	1.0000	1.0000	1.0000	DINA
Item 86	1.0000	1.0000	0.9705	1.0000	1.0000	DINA
Item 87	0.9993	0.9991	0.9878	0.9914	0.9867	DINO
Item 89	0.9903	0.9906	0.9735	0.9741	0.9697	DINO
Item 92	0.9691	0.9966	1.0000	1.0000	0.9925	DINO
Item 93	0.9253	0.8156	0.9866	1.0000	1.0000	DINA
Item 96	0.9987	0.9965	0.1924	1.0000	0.0051	DINA
Item 99	1.0000	1.0000	0.9857	1.0000	0.7902	DINA

An absolute fit test was then conducted to compare the fit of the models to the data. The *modelfit()* function was used to measure the M_2 statistic, the root mean square error of approximation ($RMSEA_2$), and the standardized root mean square residual (SRMSR). Additionally, the *itemfit()* function was employed to assess the maximum *z*-scores for both transformed correlation and log odds ratio. In these indices, lower values are preferable as they signify a

reduced discrepancy between the model predictions and the actual observed data. In the preliminary analysis (refer to Appendix B for the results), it is notable that all $RMSEA_2$ values were above 0.045, which indicates that all models displayed a poor fit. Although there is no universally accepted cutoff (Davier & Lee, 2019), several studies have suggested that an RMSEA below 0.045 denotes a good model fit (e.g., Delafontaine et al., 2022; Shi et al., 2021). Consequently, 19 items that exhibited inadequate performance in terms of the item discrimination index (IDI) and the G-DINA discrimination index (GDI) were removed. This elimination focused on items that made a minimal contribution to distinguishing between latent classes or those with low IDI and GDI scores. The removal of these poorly performing items was critical to enhancing the overall accuracy of the model. Following this refinement, Table 3 presents improved results, evidenced by lower values across various indices.

Table 3. Absolute Model Fit Indices

Model	df	M_2	$RMSEA_2$ (CI)	SRMSR	Max.z(*r*)	Max.z(*l*)
G-DINA	2486	5568.106	0.0464 (0.0447 - 0.0480)	0.0837	11.8611	12.3432
DINA	3128	12098.120	0.0705 (0.0692 - 0.0718)	0.0813	13.6626	13.9054
DINO	3128	12091.610	0.0705 (0.0691 - 0.0718)	0.0823	13.5415	13.7789
RRUM	3017	6551.055	0.0451 (0.0436 - 0.0465)	0.0831	10.1394	10.1633
LLM	3017	6489.424	0.0447 (0.0432 - 0.0462)	0.0822	10.1137	10.1389
ACDM	3017	6567.468	0.0452 (0.0437 - 0.0466)	0.0883	10.1528	10.1736
Rasch	2512	5716.850	0.0470 (0.0454 - 0.0486)	0.0842	10.1097	10.5211
1PL	2511	5665.227	0.0467 (0.0450 - 0.0483)	0.0837	10.7157	11.1573
2PL	2507	5546.834	0.0458 (0.0442 - 0.0475)	0.0842	11.0002	11.4603
MIXED	2821	7677.886	0.0546 (0.0532 - 0.0561)	0.0907	10.0860	10.1093

Note: df = Degrees of Freedom; RMSEA = Root Mean Square Error of Approximation; CI = Confidence Interval; SRMSR = Standardized Root Mean Square Residual; Max.*z*(*r*) = maximum *z* score for transformed correlation; Max.*z*(l) = maximum *z* score for log odds ratio.

To answer RQ1, a relative fit test was performed to objectively compare and find the best fitting model among the saturated (G-DINA), reduced (DINA, DINO, ACDM, LLM, & RRUM), higher-order (Rasch, 1PL, and 2PL), and a combination of different models (MIXED). Using the anova() function in the G-DINA framework, the results of the information criteria and likelihood ratio test are shown in Table 4. Based on these results, the DINA, DINO, and MIXED models were rejected ($p < 0.001$), and only the RRUM, LLM, and ACDM fitted the data well ($p = 1$). Among these potential models, the LLM demonstrated the best fit (AIC = 49819.33; BIC = 51144.11). Both AIC and BIC serve as a measure for comparing model fit, with lower values signifying a more favorable model fit (Shi et al., 2021). The consistency between the relative and absolute fit tests is evident, as the LLM not only emerges as the best model in the relative fit test but also excels in the absolute fit test by exhibiting the lowest RMSEA2 value (0.0447). Most importantly, this is the only model with a good fit (RMSEA2 < 0.045) to the data.

Table 4. Relative Model Fit Indices

Model	#par	loglik	Deviance	AIC	BIC	CAIC	SABIC	chisq	df	p-value
G-DINA	835	-24525.45	49050.90	50720.90	54359.70	55194.70	51708.92			
DINA	193	-25408.89	50817.79	51203.79	52044.85	52237.85	51432.15	1766.88	642	<0.001
DINO	193	-25376.54	50753.08	51139.08	51980.15	52173.15	51367.45	1702.18	642	<0.001
RRUM	304	-24629.31	49258.62	49866.62	51191.40	51495.40	50226.33	207.72	531	1
LLM	304	-24605.66	49211.33	49819.33	51144.11	51448.11	50179.03	160.42	531	1
ACDM	304	-24708.64	49417.28	50025.28	51350.07	51654.07	50384.99	366.38	531	1
RASCH	809	-24349.04	48698.09	50316.09	53841.58	54650.58	51273.34	-352.81		
1PL	810	-24355.82	48711.64	50331.64	53861.50	54671.50	51290.08	-339.26		
2PL	814	-24307.13	48614.26	50242.26	53789.55	54603.55	51205.43	-436.64		
MIXED	264	-31235.97	62471.94	62999.94	64150.41	64414.41	63312.32	13421.04	571	<0.001

Note: In Likelihood Ratio tests, models were tested against GDINA. #par = Number of Parameters; loglike = Log-Likelihood; AIC = Akaike Information Criterion; BIC = Bayesian Information Criterion; CIC = Consistent AIC; SABIC = Sample-Size Adjusted BIC; chisq = Chi-Square Statistic; df = Degrees of Freedom.

Consequently, the LLM was selected and used to compute the classification accuracy of the test, as well as to estimate the discrimination index and attribute prevalence. This approach supports the practice of empirically simplifying the G-DINA model, as reduced models can provide better classification results when used appropriately (Ma et al., 2016). The analysis revealed that using LLM in analyzing the dataset achieved an overall attribute profile classification accuracy of 0.8484, which indicates that it could reliably classify 84.84% of students in terms of their skill mastery of computer programming. At the attribute level, the selected model can correctly identify theoretical understanding at 97.41%, language proficiency at 99.50%, logical reasoning at 96.69%, algorithmic thinking at 90.01%, and code tracing at 94.48%. These results indicate high classification accuracy at the profile and attribute levels, demonstrating the robustness of the selected model in assessing diverse skill sets.

RQ2: What are the attributes mastered by students at the grade and individual levels?

In addressing RQ2, the attribute prevalence was analyzed to determine the proportion of students who have mastered or not mastered each attribute in computer programming (see Figure 2). According to the grade-level analysis, students mastered only two of the five evaluated attributes, indicating that the majority did not achieve mastery. Specifically, the data showed that a higher percentage of students demonstrated mastery in code tracing (60.02%) and language proficiency (81.76%). In contrast, there was a smaller proportion of students who achieved mastery in algorithmic thinking (48.82%), logical reasoning (44.44%), and theoretical understanding (44.76%). These contrasting results in attribute prevalence at the grade level highlight the need for a more balanced and focused approach in programming education. It is essential to ensure that all critical skills are adequately addressed and developed.

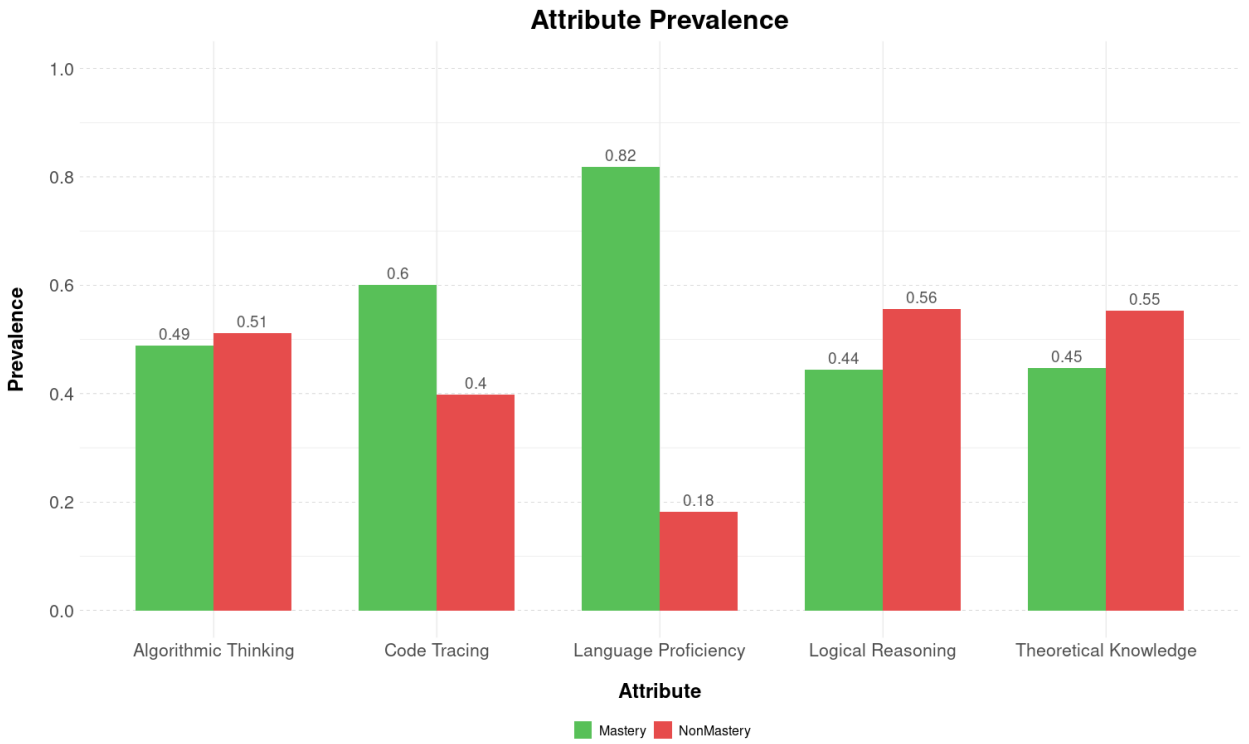
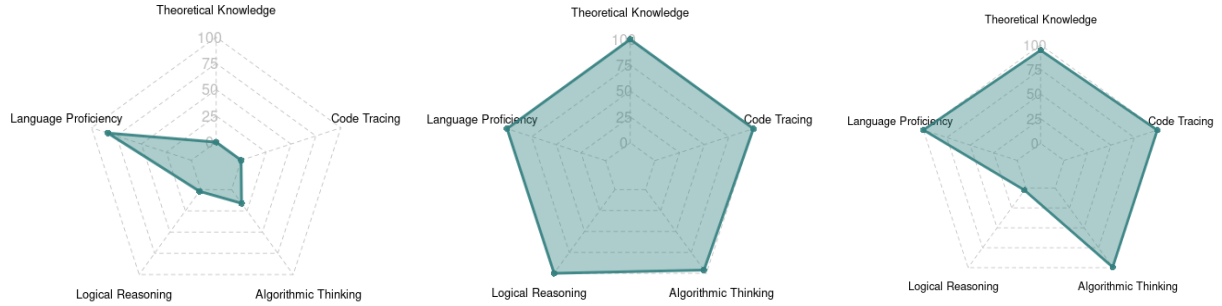


Figure 2. Attribute Prevalence of the Programming Student Population

To understand the patterns of attribute mastery among programming students, latent classes and their corresponding posterior probability percentages were analyzed. With five attributes, there are up to 32 unique attribute profiles (APs). As shown in Table 5 and Figure 3, three latent classes dominated the programming student population. These include students who mastered only language proficiency (AP 3; 16.49%; Figure 3b), those who mastered all attributes (AP 32; 15.66%; Figure 3c), and those students who mastered all attributes except logical reasoning (AP 29; 10.83%; Figure 3a). These mastery patterns are followed by 18 more latent classes that had a non-zero posterior probability. There were also 11 latent classes where the posterior probability was 0%, indicating no students fell into these categories. Overall, this individual-level analysis conforms with the results of the grade-level analysis.



(a) Mastery Pattern 01000 (b) Mastery Pattern 11111 (c) Mastery Pattern 11011

Figure 3. Individual Skill Profiles Based on Dominant Mastery Pattern

Table 5. Latent Class and Posterior Probability Percentage (Whole Population)

Attribute Profile	Attributes					Posterior Probability
	Theoretical Understanding	Language Proficiency	Logical Reasoning	Algorithmic Thinking	Code Tracing	
1	0	0	0	0	0	0.58
2	1	0	0	0	0	0.00
3	0	1	0	0	0	16.49
4	0	0	1	0	0	0.00
5	0	0	0	1	0	4.71
6	0	0	0	0	1	4.56
7	1	1	0	0	0	0.03
8	1	0	1	0	0	0.00
9	1	0	0	1	0	0.00
10	1	0	0	0	1	0.00
11	0	1	1	0	0	6.36
12	0	1	0	1	0	6.75
13	0	1	0	0	1	7.18
14	0	0	1	1	0	0.56
15	0	0	1	0	1	0.97
16	0	0	0	1	1	0.25
17	1	1	1	0	0	0.00
18	1	1	0	1	0	4.17
19	1	1	0	0	1	0.01
20	1	0	1	1	0	0.00
21	1	0	1	0	1	1.69
22	1	0	0	1	1	0.00
23	0	1	1	1	0	0.00
24	0	1	1	0	1	6.21
25	0	1	0	1	1	0.00
26	0	0	1	1	1	0.00
27	1	1	1	1	0	0.33
28	1	1	1	0	1	7.11
29	1	1	0	1	1	10.83
30	1	0	1	1	1	4.93

31	0	1	1	1	1	0.63
32	1	1	1	1	1	15.66

Note: 1 = attribute present; 0 = attribute absent

RQ3: How do these mastery profiles vary between CS and IT students?

In addressing RQ3, the proportion of IT and CS students who have or have not mastered each programming attribute was compared by activating the multi-group analysis in the GDINA library. Both relative and absolute fit tests demonstrate that the LLM is still the best model for this analysis. The grade-level analysis revealed that CS students achieved higher mastery levels than IT students in four out of five attributes. As shown in Figure 4, the data indicate that CS students demonstrated better mastery in algorithmic thinking (64.05%), language proficiency (98.70%), logical reasoning (76.11%), and theoretical understanding (61.33%). Only in code tracing (31.57%) did the IT students exhibit better mastery. Nonetheless, the mastery levels of both groups did not reach or exceed the 50% threshold in this attribute. This finding suggests that code tracing is the area where both groups show the most significant weakness.

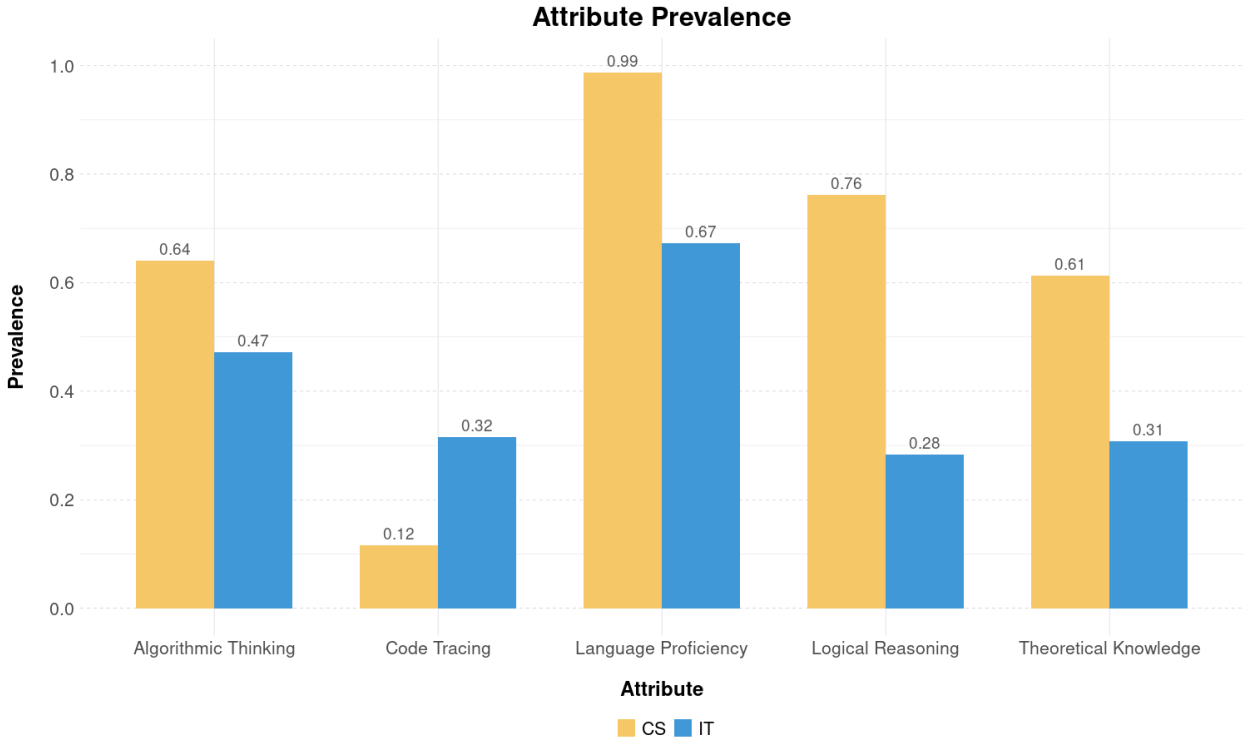


Figure 4. Attribute Prevalence Between IT and CS Programming Students

Building on the findings from the multi-group analysis, the individual-level analysis further explores the differences in the patterns of attribute mastery between CS and IT students. It is not surprising to observe distinct mastery profiles within these groups (see Table 6), given the varied learning outcomes previously noted. For example, the CS student population is dominated by students who have mastered all attributes except algorithmic thinking (AP 28; 28.15%).

Conversely, the IT student population is dominated by students who have mastered only language proficiency (AP 3; 34.06%). Although the CS group exhibits greater mastery in this attribute, it still represents the attribute most frequently mastered by the IT group. This individual-level analysis aligns with the overarching theme from our broader analysis: distinct educational trajectories are evident within the IT and CS student populations.

Table 6. Latent Class and Posterior Probability Percentage (IT vs CS)

Attribute Profile	Attributes					Posterior Probability	
	Theoretical Understanding	Language Proficiency	Logical Reasoning	Algorithmic Thinking	Code Tracing	IT	CS
1	0	0	0	0	0	0.00	0.00
2	1	0	0	0	0	0.00	0.00
3	0	1	0	0	0	34.06	3.50
4	0	0	1	0	0	0.00	1.11
5	0	0	0	1	0	9.79	0.33
6	0	0	0	0	1	0.48	0.00
7	1	1	0	0	0	0.00	1.14
8	1	0	1	0	0	0.00	0.00
9	1	0	0	1	0	0.00	0.00
10	1	0	0	0	1	0.00	0.00
11	0	1	1	0	0	0.00	18.15
12	0	1	0	1	0	0.00	0.00
13	0	1	0	0	1	11.63	4.38
14	0	0	1	1	0	1.62	0.19
15	0	0	1	0	1	0.00	0.00
16	0	0	0	1	1	3.13	0.00
17	1	1	1	0	0	0.00	0.00
18	1	1	0	1	0	0.77	0.00
19	1	1	0	0	1	9.02	7.43
20	1	0	1	1	0	0.00	0.00
21	1	0	1	0	1	0.00	0.00
22	1	0	0	1	1	0.00	0.00
23	0	1	1	1	0	0.00	0.80
24	0	1	1	0	1	0.48	9.95
25	0	1	0	1	1	0.00	0.00

26	0	0	1	1	1	7.26	0.00
27	1	1	1	1	0	0.00	0.69
28	1	1	1	0	1	0.23	28.15
29	1	1	0	1	1	9.22	6.71
30	1	0	1	1	1	11.82	0.00
31	0	1	1	1	1	0.26	0.42
32	1	1	1	1	1	0.20	17.04

Note: 1 = attribute present; 0 = attribute absent; IT = Information Technology; CS = Computer Science

DISCUSSION

High student dropout rates in programming courses are a significant issue, often resulting in lower overall graduation rates. This barrier not only hinders the development of a skilled workforce capable of meeting the evolving demands of various sectors but also limits the ability of individuals to engage with and contribute to the digital economy. Prior works have established that programming students face various learning difficulties (Garcia, 2021). However, the exact nature of their weaknesses remains vague, posing a challenge in determining which specific programming-related attributes students lack. This gap exists mainly because most studies on programming education focus on evaluating overall academic performance. Recognizing the importance of pinpointing specific skill mastery profiles, this study applied CDM to assess the cognitive abilities of programming students. By identifying specific areas where a student may struggle or excel in programming, this study offers empirical evidence that can assist educators in customizing their instructional approaches. Moreover, it can inform the development of curricula to better cater to the varied cognitive requirements of students in programming courses.

Model fit analyses were initially conducted to ensure the accuracy of the diagnostic conclusions derived from the data. The LLM demonstrated the best fit as evidenced by both relative and absolute fit indices. This selection also supports the empirical simplification of saturated models, as reduced models can provide better classification rates than their more complex counterparts (Ma et al., 2016). Based on the data analyses, the LLM exhibited high accuracy in both profile and attribute-level classification. Using this model, the attribute prevalence was then analyzed to determine the mastery levels of each programming attribute. Unfortunately, the grade-level analysis revealed that programming students only mastered language proficiency and code tracing. These two attributes indicate that students have a fundamental understanding of the syntax and semantics of programming languages, as well as the ability to follow and predict the execution flow of programs. Mastery in these areas is essential for basic programming tasks, but it does not cover the full spectrum of skills required for more advanced problem-solving and program development (Graafsma et al., 2023). The reliance on language proficiency and code tracing skills alone may result in a limited ability to tackle complex programming challenges, which require a deeper understanding of underlying principles and the ability to apply logical and algorithmic thinking (Garcia et al., 2023). This limited skill set suggests a significant gap in the educational approach that may need to be addressed to ensure students develop a more comprehensive and well-rounded programming skill set.

While they demonstrate an understanding of the programming language and can follow program execution, students lack mastery in algorithmic thinking, logical reasoning, and theoretical understanding. These deficiencies are particularly concerning because effective program development necessitates not just theoretical understanding but also mastery of algorithmic thinking and logical reasoning (Thuné & Eckerdal, 2019). Without strong skills in these areas, students may struggle to design efficient algorithms, understand the abstract concepts underlying programming tasks, and apply logical frameworks to problem-solving (Garcia, Juanatas, et al., 2022). The lack of algorithmic thinking and logical reasoning skills can also impede students' ability to debug and optimize their code effectively, which can lead to persistent errors and inefficient solutions. Graafsma et al. (2023) also found that logical reasoning has been shown to predict academic success in computer programming courses. Unfortunately, the absence of these fundamental programming skills at the earliest stage of their academic program poses a significant threat to their long-term success (Xie et al., 2019). Students may find themselves struggling to tackle the more intricate and abstract aspects of programming. As they face challenges in meeting course expectations and in dealing with complex programming tasks, their confidence might decrease (Kovari & Katona, 2023). This diminished confidence may lead them to question their decision to pursue programming-related studies.

From a practical standpoint, this finding illuminates what contributes to academic failures and eventual dropout in programming education. Garcia and Revano (2021) asserted that students who lack a firm grasp of theoretical understanding may struggle with more advanced programming topics. This difficulty can result in students facing increasing challenges as they progress through their coursework. Unfortunately, it often translates into poor academic performance (Hota et al., 2023), which can lead to a cycle of frustration and discouragement. This emotional response can further hinder their learning process and motivation, exacerbating their struggles in the field of programming. This assertion is supported by empirical evidence demonstrating that fun plays a vital role in coding, as it has a positive and indirect effect on learning (Tisza & Markopoulos, 2021). Furthermore, the absence of algorithmic thinking (Lamagna, 2015) and logical reasoning (Barlow-Jones & van der Westhuizen, 2017; Graafsma et al., 2023) may present significant obstacles for students when tackling real-world programming challenges. Without these fundamental skills, students may find themselves struggling to effectively utilize their language proficiency in practical scenarios. This finding becomes even more problematic considering that individual-level analysis reveals that the majority of the programming student population excels only in language proficiency. Unfortunately, there is a significant difference between merely knowing a programming language and being able to effectively use it, a skill that requires a mastery of both algorithmic thinking and logical reasoning. Students who do not master these fundamental skills often lack confidence in their programming abilities. This lack of self-assurance can contribute to performance anxiety and the decision to drop out (Dirzyte et al., 2023; Garcia, 2023). Recognizing their deficiencies in essential programming skills, some students may opt to pursue alternative career paths.

When analyzing the student population separately by academic program, distinct learning outcomes emerged between IT and CS programming students. Despite having access to identical educational resources and programming curriculum, CS students displayed superior mastery compared to their IT counterparts. This disparity suggests that factors other than shared

educational content contribute to these differences. Unfortunately, there is a paucity of literature on this subject, and this study represents the first to compare the programming skill mastery profiles of IT and CS students. Future research should aim to identify the reasons behind these differences to determine the precise interventions necessary to improve programming education. In this context, a potential explanation could be the curriculum. From a curricular standpoint, CS programs often focus on theoretical foundations, algorithmic thinking, and problem-solving skills, which are essential for understanding complex computing concepts. Conversely, IT programs might place more emphasis on practical, application-oriented skills. This divergence in curricular focus could result in CS students developing a deeper understanding of computational theory and algorithms. In such a scenario, it could be anticipated that a more pronounced delineation in skill mastery profiles will manifest as students advance through their academic curriculum. Consequently, longitudinal assessments are necessary to explain these developmental trajectories in future research. The implication of this finding also underscores the importance of continuous curriculum evaluation and the adoption of adaptive teaching methods in programming education. By doing so, educators can ensure that students not only gain comprehensive programming knowledge but also develop a well-rounded set of skills essential for their future careers (Schnieder & Williams, 2022). Overall, these differences underscore the need for tailored educational strategies that cater to the unique strengths and weaknesses of each student group. Recognizing and addressing these differences could lead to more effective and personalized programming education, thereby optimizing learning outcomes for both IT and CS students.

The study needs to acknowledge limitations that can provide avenues for future research. While failure and dropout rates are a significant motivation for this research, the primary focus was to identify specific cognitive strengths and weaknesses of students in introductory programming courses. Linking these cognitive profiles to actual dropout rates is essential for developing targeted interventions and improving student retention strategies in programming courses. This connection, however, requires a broader, more longitudinal approach that could be explored in future research. Additionally, the study focused on specific cognitive attributes present in the examination analyzed. However, there are other essential characteristics that programming students should possess, such as debugging skills and mathematical abilities (Graafsma et al., 2023; Schnieder & Williams, 2022). Future research should aim to include a broader range of cognitive attributes to provide a more comprehensive diagnostic profile of programming students. In addition, the assessment used in this study was in a multiple-choice format. While multiple-choice questions can effectively evaluate certain types of knowledge and skills, other assessment approaches (e.g., project-based assessments, coding assignments, and peer reviews) may be more suitable for capturing the full range of cognitive abilities required in programming (Garcia, 2023; Nakayama et al., 2021). These alternative formats could offer deeper insights into students' practical skills and application of knowledge in real-world scenarios. Lastly, the generalizability of the findings may be limited due to variations in curricula across different institutions. Different programming courses may emphasize various aspects of programming skills, which could affect students' cognitive profiles and their development of specific abilities. Future research should consider these curriculum differences to ensure a more comprehensive understanding of programming education across diverse educational contexts. Addressing these limitations in future studies can build upon the findings to create a more holistic understanding of the factors influencing success and retention in programming education.

CONCLUSION

Computer programmers are currently in high demand due to ongoing digitalization and the increasing complexity of technological solutions across various industries. It is imperative that education systems continuously develop a skilled workforce capable of meeting the evolving demands of these sectors. While previous studies have identified learning difficulties contributing to dropout rates among programming students, the specific nature of these weaknesses has remained unclear. To address this gap, this study employed a CDM approach to assess the skill mastery profiles of programming students. The findings revealed that students exhibited proficiency primarily in code tracing and language proficiency but lacked mastery in theoretical understanding, logical reasoning, and algorithmic thinking. The student population was predominantly categorized into three latent classes: those students proficient in all areas except logical reasoning, those students proficient only in language proficiency, and those students proficient in all assessed attributes. A comparison between students based on their academic programs showed that CS students generally outperformed IT students in logical reasoning, algorithmic thinking, language proficiency, and theoretical understanding. Conversely, IT students excelled better in code tracing. This pattern differed from the dominant latent classes within each group, with CS students mostly mastering all attributes except algorithmic thinking and IT students predominantly mastering language proficiency. These findings highlight a notable discrepancy in mastery levels across various programming skills, which has significant implications for curriculum development and instructional strategies.

In conclusion, this study provides a critical first step in understanding the cognitive skill profiles of introductory programming students. By systematically identifying the specific areas where students excel and struggle, this research offers invaluable insights that can be leveraged to enhance educational strategies and outcomes. Programming instructors can use these insights to design targeted interventions that address specific cognitive weaknesses, thereby improving student outcomes and potentially reducing dropout rates. Furthermore, the implications of this study extend beyond immediate educational settings. By highlighting the importance of tailored educational strategies, this research underscores the need for continuous adaptation and improvement in teaching practices to meet the dynamic requirements of the tech industry. The insights gained from this study can inform policy decisions and curriculum design at institutional and broader educational levels, which may contribute to the development of a more competent and adaptable workforce. This study also advances the field of educational research by demonstrating the utility and effectiveness of CDM in providing actionable diagnostic feedback. The application of CDM in this context not only enriches the existing body of literature on programming education but also sets a precedent for its use in other educational domains. By showcasing how CDM can uncover cognitive profiles, this research paves the way for future studies to implement similar diagnostic approaches across various disciplines. In essence, this study not only contributes to the immediate improvement of programming education but also lays the groundwork for a more profound understanding of cognitive skill development and its implications. By bridging the gap between cognitive diagnostics and practical educational strategies, this research fosters a more supportive and effective learning environment.

REFERENCES

Agbo, F. J., Oyelere, S. S., Suhonen, J., & Adewumi, S. (2019). A Systematic Review of Computational Thinking Approach for Programming Education in Higher Education Institutions. *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*. https://doi.org/10.1145/3364510.3364521

Akaike, H. (1974). A New Look at the Statistical Model Identification. *IEEE Transactions on Automatic Control*, *19*(6), 716-723. https://doi.org/10.1109/TAC.1974.1100705

Angeli, C. (2022). The Effects of Scaffolded Programming Scripts on Pre-Service Teachers' Computational Thinking: Developing Algorithmic Thinking Through Programming Robots. *International Journal of Child-Computer Interaction*, *31*, 1-20. https://doi.org/10.1016/j.ijcci.2021.100329

Asian Development Bank. (2022). *Digital Jobs and Digital Skills: A Shifting Landscape in Asia and the Pacific*.

Barlow-Jones, G., & van der Westhuizen, D. (2017). Problem Solving as a Predictor of Programming Performance. *ICT Education*, 209-216. https://doi.org/10.1007/978-3-319-69670-6_14

Chandía, E., Sanhueza, T., Mansilla, A., Morales, H., Huencho, A., & Cerda, G. (2023). Nonparametric Cognitive Diagnosis of Profiles of Mathematical Knowledge of Teacher Education Candidates. *Current Psychology*, *42*(36), 32498-32511. https://doi.org/10.1007/s12144-023-04256-2

Chen, J., de la Torre, J., & Zhang, Z. (2013). Relative and Absolute Fit Evaluation in Cognitive Diagnosis Modeling. *Journal of Educational Measurement*, *50*(2), 123-140. https://doi.org/10.1111/j.1745-3984.2012.00185.x

Davier, M. v., & Lee, Y.-S. (2019). *Handbook of Diagnostic Classification Models: Models and Model Extensions, Applications, Software Packages*. Springer. https://doi.org/10.1007/978-3-030-05584-4

de la Torre, J., & Chiu, C.-Y. (2016). A General Method of Empirical Q-matrix Validation. *Psychometrika*, *81*(2), 253-273. https://doi.org/10.1007/s11336-015-9467-8

de la Torre, J., & Lee, Y.-S. (2013). Evaluating the Wald Test for Item-Level Comparison of Saturated and Reduced Models in Cognitive Diagnosis. *Journal of Educational Measurement*, *50*(4), 355-373. https://doi.org/https://doi.org/10.1111/jedm.12022

de la Torre, J., & Minchen, N. (2014). Cognitively Diagnostic Assessments and the Cognitive Diagnosis Model Framework. *Psicología Educativa*, *20*, 89-97. https://doi.org/10.1016/j.pse.2014.11.001

de la Torre, J. (2011). The Generalized DINA Model Framework. *Psychometrika*, *76*(2), 179-199. https://doi.org/10.1007/s11336-011-9207-7

Delafontaine, J., Chen, C., Park, J. Y., & Van den Noortgate, W. (2022). Using Country-Specific Q-Matrices for Cognitive Diagnostic Assessments with International Large-Scale Data. *Large-scale Assessments in Education*, *10*(1), 1-36. https://doi.org/10.1186/s40536-022-00138-4

Dirzyte, A., Perminas, A., Kaminskis, L., Žebrauskas, G., Sederevičiūtė – Pačiauskienė, Ž., Šliogerienė, J., Suchanova, J., Rimašiūtė – Knabikienė, R., Patapas, A., & Gajdosikiene, I. (2023). Factors Contributing to Dropping Out of Adults' Programming E-Learning. *Heliyon*, *9*(12), 1-16. https://doi.org/10.1016/j.heliyon.2023.e22113

Djurdjevic-Pahl, A., Pahl, C., Fronza, I., & El Ioini, N. (2017). A Pathway into Computational Thinking in Primary Schools. *Emerging Technologies for Education*, 165-175. https://doi.org/10.1007/978-3-319-52836-6_19

Effatpanah, F., Baghaei, P., & Boori, A. A. (2019). Diagnosing EFL Learners' Writing Ability: A Diagnostic Classification Modeling Analysis. *Language Testing in Asia*, *9*(1), 1-23. https://doi.org/10.1186/s40468-019-0090-y

European Labour Authority. (2023). *Report on Labour Shortages and Surpluses – 2022*. Publications Office of the European Union. https://doi.org/10.2883/50704

Garcia, M. B. (2021). Cooperative Learning in Computer Programming: A Quasi-Experimental Evaluation of Jigsaw Teaching Strategy with Novice Programmers. *Education and Information Technologies*, *26*(4), 4839-4856. https://doi.org/10.1007/s10639-021-10502-6

Garcia, M. B. (2023). Facilitating Group Learning Using an Apprenticeship Model: Which Master is More Effective in Programming Instruction? *Journal of Educational Computing Research*, *61*(6), 1207-1231. https://doi.org/10.1177/07356331231170382

Garcia, M. B., Enriquez, J. B. R., Adao, R. T., & Happonen, A. (2022). "Hey IDE, Display Hello World": Integrating a Voice Coding Approach in Hands-on Computer Programming Activities. *2022 IEEE 14th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*, 1-6. https://doi.org/10.1109/HNICEM57413.2022.10109412

Garcia, M. B., Juanatas, I. C., & Juanatas, R. A. (2022). TikTok as a Knowledge Source for Programming Learners: A New Form of Nanolearning? *2022 10th International Conference on Information and Education Technology (ICIET)*, 219-223. https://doi.org/10.1109/ICIET55102.2022.9779004

Garcia, M. B., & Revano, T. F. (2021). Assessing the Role of Python Programming Gamified Course on Students' Knowledge, Skills Performance, Attitude, and Self-Efficacy. *2021 IEEE 13th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*, 1-5. https://doi.org/10.1109/HNICEM54116.2021.9731935

Garcia, M. B., Revano, T. F., Maaliw, R. R., Lagrazon, P. G. G., Valderama, A. M. C., Happonen, A., Qureshi, B., & Yilmaz, R. (2023). Exploring Student Preference between AI-Powered ChatGPT and Human-Curated Stack Overflow in Resolving Programming Problems and Queries. *2023 IEEE 15th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*, 1-6. https://doi.org/10.1109/HNICEM60674.2023.10589162

Graafsma, I. L., Robidoux, S., Nickels, L., Roberts, M., Polito, V., Zhu, J. D., & Marinus, E. (2023). The Cognition of Programming: Logical Reasoning, Algebra and Vocabulary Skills Predict Programming Performance Following an Introductory Computing Course. *Journal of Cognitive Psychology*, *35*(3), 364-381. https://doi.org/10.1080/20445911.2023.2166054

Guo, P. J. (2018). Non-Native English Speakers Learning Computer Programming: Barriers, Desires, and Design Opportunities. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3173574.3173970

Hartz, S. M. (2002). A Bayesian Framework for the Unified Model for Assessing Cognitive Abilities: Blending Theory with Practicality. *Dissertation Abstracts International: Section B: The Sciences and Engineering*, *63*(2-B), 864. https://psycnet.apa.org/record/2002-95016-234

Helm, C., Warwas, J., & Schirmer, H. (2022). Cognitive Diagnosis Models of Students' Skill Profiles as a Basis for Adaptive Teaching: An Example From Introductory Accounting Classes. *Empirical Research in Vocational Education and Training*, *14*(1), 1-30. https://doi.org/10.1186/s40461-022-00137-3

Hota, C. P. P. K., Asanambigai, V., & Lakshmi, D. (2023). Predicting Academic Grades of Students in Computer Programming Using Classification Algorithms. *2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 607-612. https://doi.org/10.1109/ICACCS57279.2023.10112996

International Labour Organization. (2020). *Skills Shortages and Labour Migration in the Field of Information and Communication Technology in Canada, China, Germany and Singapore*. https://www.ilo.org/wcmsp5/groups/public/—ed_dialogue/—sector/documents/publication/wcms_755663.pdf

International Labour Organization. (2021). *Changing Demand for Skills in Digital Economies and Societies: Literature Review and Case Studies from Low- and Middle-Income Countries*. https://www.ilo.org/wcmsp5/groups/public/-–ed_emp/—ifp_skills/documents/publication/wcms_831372.pdf

Jang, E. E., Dunlop, M., Park, G., & van der Boom, E. H. (2015). How Do Young Students With Different Profiles of Reading Skill Mastery, Perceived Ability, and Goal Orientation Respond to Holistic Diagnostic Feedback? *Language Testing*, *32*(3), 359-383. https://doi.org/10.1177/0265532215570924

Junker, B. W., & Sijtsma, K. (2001). Cognitive Assessment Models with Few Assumptions, and Connections with Nonparametric Item Response Theory. *Applied Psychological Measurement*, *25*(3), 258-272. https://doi.org/10.1177/01466210122032064

Kiss, G., & Arki, Z. (2017). The Influence of Game-based Programming Education on the Algorithmic Thinking. *Procedia - Social and Behavioral Sciences*, *237*, 613-617. https://doi.org/10.1016/j.sbspro.2017.02.020

Kovari, A., & Katona, J. (2023). Effect of Software Development Course on Programming Self-Efficacy. *Education and Information Technologies*, *28*(9), 10937-10963. https://doi.org/10.1007/s10639-023-11617-8

Kumar, A. N. (2015). Solving Code-tracing Problems and its Effect on Code-writing Skills Pertaining to Program Semantics. *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, 314–319. https://doi.org/10.1145/2729094.2742587

Lamagna, E. A. (2015). Algorithmic Thinking Unplugged. *Journal of Computing Sciences in Colleges*, *30*(6), 45-52. https://dl.acm.org/doi/10.5555/2753024.2753036

Lee, Y.-S., de la Torre, J., & Park, Y. S. (2012). Relationships Between Cognitive Diagnosis, CTT, and IRT Indices: An Empirical Investigation. *Asia Pacific Education Review*, *13*(2), 333-345. https://doi.org/10.1007/s12564-011-9196-3

Li, H., Hunter, C. V., & Lei, P.-W. (2015). The Selection of Cognitive Diagnostic Models for a Reading Comprehension Test. *Language Testing*, *33*(3), 391-409. https://doi.org/10.1177/0265532215590848
Li, T., & Traynor, A. (2022). The Use of Cognitive Diagnostic Modeling in the Assessment of Computational Thinking. *AERA Open*, *8*, 23328584221081256. https://doi.org/10.1177/23328584221081256
Li, Y., Zhen, M., & Liu, J. (2021). Validating a Reading Assessment Within the Cognitive Diagnostic Assessment Framework: Q-Matrix Construction and Model Comparisons for Different Primary Grades. *Frontiers in Psychology*, *12*, 1-13. https://doi.org/10.3389/fpsyg.2021.786612
Lira, C. D., Wong, R., & Adesope, O. (2022). A Systematic Review on the Effectiveness of Programming Camps on Middle School Students' Programming Knowledge and Attitudes of Computing. *Journal of Computing Sciences in Colleges*, *38*(1), 89-98. https://dl.acm.org/doi/abs/10.5555/3575618.3575627
Liu, Y., Zhang, T., Wang, X., Yu, G., & Li, T. (2023). New Development of Cognitive Diagnosis Models. *Frontiers of Computer Science*, *17*(1), 1-13. https://doi.org/10.1007/s11704-022-1128-3
Ma, W., & de la Torre, J. (2020a). An Empirical Q-Matrix Validation Method for the Sequential Generalized DINA Model. *British Journal of Mathematical and Statistical Psychology*, *73*(1), 142-163. https://doi.org/10.1111/bmsp.12156
Ma, W., & de la Torre, J. (2020b). GDINA: An R Package for Cognitive Diagnosis Modeling. *Journal of Statistical Software*, *93*(14), 1-26. https://doi.org/10.18637/jss.v093.i14
Ma, W., Iaconangelo, C., & de la Torre, J. (2016). Model Similarity, Model Selection, and Attribute Classification. *Applied Psychological Measurement*, *40*(3), 200-217. https://doi.org/10.1177/0146621615621717
Macrides, E., Miliou, O., & Angeli, C. (2022). Programming in Early Childhood Education: A Systematic Review. *International Journal of Child-Computer Interaction*, *32*, 1-17. https://doi.org/10.1016/j.ijcci.2021.100396
Maris, E. (1999). Estimating Multiple Classification Latent Class Models. *Psychometrika*, *64*(2), 187-212. https://doi.org/10.1007/BF02294535
Meng, Y., Wang, Y., & Zhao, N. (2023). Cognitive Diagnostic Assessment of EFL Learners' Listening Barriers Through Incorrect Responses. *Frontiers in Psychology*, *14*, 1-11. https://doi.org/10.3389/fpsyg.2023.1126106
Nakayama, M., Uto, M., Temperini, M., & Sciarrone, F. (2021). Estimating Ability of Programming Skills Using IRT based Peer Assessments. *2021 19th International Conference on Information Technology Based Higher Education and Training (ITHET)*, 1-6. https://doi.org/10.1109/ITHET50392.2021.9759571
Ou, Q., Liang, W., He, Z., Liu, X., Yang, R., & Wu, X. (2023). Investigation and Analysis of the Current Situation of Programming Education in Primary and Secondary Schools. *Heliyon*, *9*(4), 1-16. https://doi.org/10.1016/j.heliyon.2023.e15530
Paulsen, J., & Valdivia, D. S. (2022). Examining Cognitive Diagnostic Modeling in Classroom Assessment Conditions. *The Journal of Experimental Education*, *90*(4), 916-933. https://doi.org/10.1080/00220973.2021.1891008
Qayyum, N. u., Seman, M. S. A., Shah, A., Qureshi, M. S., & Raza, A. (2018). A Review of Programming Code Assessment Approaches. *2018 IEEE 5th International Conference on Engineering Technologies and Applied Sciences (ICETAS)*, 1-5. https://doi.org/10.1109/ICETAS.2018.8629221
Ravand, H., & Robitzsch, A. (2018). Cognitive Diagnostic Model of Best Choice: A Study of Reading Comprehension. *Educational Psychology*, *38*(10), 1255-1277. https://doi.org/10.1080/01443410.2018.1489524
Rupp, A. A., & Templin, J. L. (2008). Unique Characteristics of Diagnostic Classification Models: A Comprehensive Review of the Current State-of-the-Art. *Measurement: Interdisciplinary Research and Perspectives*, *6*(4), 219-262. https://doi.org/10.1080/15366360802490866
Russell, S. (2022). Automated Code Tracing Exercises for CS1. *Proceedings of 6th Conference on Computing Education Practice*, 13–16. https://doi.org/10.1145/3498343.3498347
Schnieder, M., & Williams, S. (2022). How to Assess Programming Skills: Review and Analysis. *2022 IEEE German Education Conference (GeCon)*, 1-7. https://doi.org/10.1109/GeCon55699.2022.9942789
Schwarz, G. (1978). Estimating the Dimension of a Model. *The Annals of Statistics*, *6*(2), 461-464. https://doi.org/10.1214/aos/1176344136
Shi, Q., Ma, W., Robitzsch, A., Sorrel, M. A., & Man, K. (2021). Cognitively Diagnostic Analysis Using the G-DINA Model in R. *Psych*, *3*(4), 812-835. https://doi.org/10.3390/psych3040052
Stankov, E., Jovanov, M., & Madevska Bogdanova, A. (2023). Smart Generation of Code Tracing Questions for Assessment in Introductory Programming. *Computer Applications in Engineering Education*, *31*(1), 5-25. https://doi.org/https://doi.org/10.1002/cae.22567

Tatsuoka, K. K. (1983). Rule Space: An Approach for Dealing With Misconceptions Based on Item Response Theory. *Journal of Educational Measurement*, *20*(4), 345-354. https://doi.org/10.1111/j.1745-3984.1983.tb00212.x

Templin, J. L., & Henson, R. A. (2006). Measurement of Psychological Disorders Using Cognitive Diagnosis Models. *Psychological Methods*, *11*(3), 287-305. https://doi.org/10.1037/1082-989X.11.3.287

Thuné, M., & Eckerdal, A. (2019). Analysis of Students' Learning of Computer Programming in a Computer Laboratory Context. *European Journal of Engineering Education*, *44*(5), 769-786. https://doi.org/10.1080/03043797.2018.1544609

Tisza, G., & Markopoulos, P. (2021). Understanding the Role of Fun in Learning to Code. *International Journal of Child-Computer Interaction*, *28*, 1-10. https://doi.org/10.1016/j.ijcci.2021.100270

Tsukamoto, H., Oomori, Y., Nagumo, H., Takemura, Y., Monden, A., & Matsumoto, K. i. (2017). Evaluating Algorithmic Thinking Ability of Primary Schoolchildren Who Learn Computer Programming. *2017 IEEE Frontiers in Education Conference (FIE)*, 1-8. https://doi.org/10.1109/FIE.2017.8190609

Wu, X., Sun, S., Xu, T., & Wang, A. (2024). Research on the Selection of Cognitive Diagnosis Model From the Perspective of Experts. *Current Psychology*, *43*(15), 13802-13810. https://doi.org/10.1007/s12144-023-05438-8

Xie, B., Loksa, D., Nelson, G. L., Davidson, M. J., Dong, D., Kwik, H., Tan, A. H., Hwa, L., Li, M., & Ko, A. J. (2019). A Theory of Instruction for Introductory Programming Skills. *Computer Science Education*, *29*(2-3), 205-253. https://doi.org/10.1080/08993408.2019.1565235

Zhang, Y., Jin, Y., Xiong, Z., Leung, S. O., Chen, G., Li, N., & Li, B. (2022). Personalized Assessment: Applying Higher-Order Cognitive Diagnosis Models in Secondary Mathematics. *Asian Journal for Mathematics Education*, *1*(4), 455-474. https://doi.org/10.1177/27527263221136301

Zhang, Y., Paquette, L., Pinto, J. D., & Fan, A. X. (2023). Utilizing Programming Traces to Explore and Model the Dimensions of Novices' Code-Writing Skill. *Computer Applications in Engineering Education*, *31*(4), 1041-1058. https://doi.org/https://doi.org/10.1002/cae.22622

Appendix A. Validated Q-Matrix for the CCS0003 Final Examination

Item	Attributes					Item	Attributes				
	TU	LP	LR	AT	CT		TU	LP	LR	AT	CT
1	1	0	0	0	0	51	1	1	1	1	1
2	1	0	0	0	0	52	0	1	1	1	1
3	1	0	0	0	0	53	0	1	1	1	1
4	1	0	0	0	0	54	1	0	1	1	1
5	1	0	0	0	0	55	1	0	0	0	0
6	1	0	0	0	0	56	1	0	0	0	0
7	1	0	0	0	0	57	0	1	1	1	1
8	1	0	0	0	0	58	0	1	1	1	1
9	1	0	0	0	0	59	0	1	1	1	1
10	1	0	0	0	0	60	1	0	0	0	0
11	1	0	0	0	0	61	0	1	1	1	1
12	1	0	0	0	0	62	0	1	1	1	1
13	1	1	0	0	0	63	1	1	1	1	1
14	1	0	0	0	0	64	1	1	1	1	1
15	0	1	0	0	0	65	1	1	1	1	1
16	1	0	0	0	0	66	1	0	0	0	0
17	1	0	0	0	0	67	1	1	1	1	1
18	1	0	0	0	0	68	0	1	1	1	1
19	0	1	0	0	1	69	1	1	0	0	1
20	0	1	0	0	1	70	1	0	0	0	0
21	1	0	0	0	0	71	1	0	0	0	0
22	1	0	1	0	0	72	0	1	1	1	1
23	1	0	1	0	0	73	1	1	1	1	1
24	0	0	1	0	0	74	1	0	0	0	0
25	1	1	1	1	0	75	1	1	1	1	1
26	0	1	0	0	0	76	1	0	0	0	0
27	1	0	0	0	0	77	1	1	1	1	1
28	1	0	0	0	0	78	0	1	1	1	1
29	0	1	0	0	0	79	0	1	1	1	1
30	0	1	0	0	0	80	0	1	1	1	0

31	1	0	0	0	0	81	1	1	1	1	1
32	0	1	1	0	1	82	1	0	0	0	0
33	1	0	0	0	0	83	0	1	1	1	1
34	0	1	0	0	0	84	1	1	1	1	1
35	1	0	0	0	0	85	1	1	1	1	1
36	1	0	0	0	0	86	1	1	1	1	1
37	1	0	0	0	0	87	1	1	0	0	0
38	0	0	1	0	0	88	1	0	0	0	0
39	1	0	0	0	0	89	1	1	0	0	0
40	0	1	1	0	1	90	1	0	0	0	0
41	0	1	1	0	0	91	1	0	0	0	0
42	0	1	0	0	0	92	0	1	1	1	1
43	0	1	0	0	0	93	1	1	1	1	1
44	1	0	0	0	0	94	1	0	0	0	0
45	0	0	0	0	1	95	1	0	0	0	0
46	1	0	0	0	0	96	1	1	1	1	1
47	1	0	0	0	0	97	1	0	0	0	0
48	0	1	1	1	1	98	1	0	0	0	0
49	0	1	0	0	0	99	1	1	1	1	1
50	1	0	0	0	0	100	1	0	0	0	0

Note: 1 = attribute present; 0 = attribute absent; TK = Theoretical Understanding (68 items); LP = Language Proficiency (48 items); LR = Logical Reasoning (37 items); AT = Algorithmic Thinking (30 items); CT = Code Tracing (35 items)

Appendix B. Preliminary Results of the Absolute Model Fit Indices

Model	df	M_2	$RMSEA_2$ (CI)	SRMSR	Max.$z(r)$	Max.$z(l)$
G-DINA	4133	11292.72	0.0548 (0.0536 - 0.0560)	0.0889	11.2436	11.2768
DINA	4819	19455.88	0.0726 (0.0715 - 0.0736)	0.0835	13.6285	13.8752
DINO	4819	19565.19	0.0728 (0.0718 - 0.0739)	0.0844	13.3964	13.6361
RRUM	4701	11741.25	0.0509 (0.0498 - 0.0521)	0.0882	11.9138	11.9985
LLM	4701	12973.44	0.0552 (0.0541 - 0.0564)	0.0834	10.7854	10.8180
ACDM	4701	11324.77	0.0494 (0.0483 - 0.0506)	0.0883	11.3965	11.8593
RASCH	4159	13151.21	0.0612 (0.0600 - 0.0624)	0.0848	9.5531	9.9198
1PL	4158	12841.23	0.0602 (0.0590 - 0.0613)	0.0847	10.4605	10.5157
2PL	4154	12982.16	0.0607 (0.0595 - 0.0619)	0.0837	10.1054	10.5014
MIXED	4786	16532.49	0.0652 (0.0641 - 0.0663)	0.0873	13.0427	13.2494

Note: df = Degrees of Freedom; CI = Confidence Interval; RMSEA = Root Mean Square Error of Approximation; SRMSR = Standardized Root Mean Square Residual; Max.$z(r)$ = maximum z score for transformed correlation; Max.z(l) = maximum z score for log odds ratio.

RELATED RESEARCH

Conference Paper

"*Hey IDE, Display Hello World*": Integrating a Voice Coding Approach in Hands-on Computer Programming Activities

Manuel B. Garcia, John Benedic R. Enriquez, Rossana T. Adao, and Ari Happonen (2022). *2022 IEEE 14th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*. https://manuelgarcia.info/publication/voice-coding-approach

Research Article

Facilitating Group Learning Using an Apprenticeship Model: Which Master is More Effective in Programming Instruction?

Manuel B. Garcia (2023). *Journal of Educational Computing Research*. https://manuelgarcia.info/publication/group-learning-programming

Conference Paper

TikTok as a Knowledge Source for Programming Learners: A New Form of Nanolearning?

Manuel B. Garcia, Irish C. Juanatas, and Roben A. Juanatas (2022). *2022 10th International Conference on Information and Education Technology (ICIET)*. https://manuelgarcia.info/publication/tiktok-programming-learners

LET'S COLLABORATE!

If you are looking for research collaborators, please do not hesitate to contact me at mbgarcia@feutech.edu.ph.

ABOUT THE CORRESPONDING AUTHOR:

Manuel B. Garcia is a professor of information technology and the founding director of the Educational Innovation and Technology Hub (EdITH) at FEU Institute of Technology, Manila, Philippines. His interdisciplinary research interests include topics that, individually or collectively, span the disciplines of education and information technology. He is a licensed professional teacher and a proud member of the *National Research Council of the Philippines* – an attached agency to the country's Department of Science and Technology (DOST-NRCP).