Assessment of Algorithmic Abstraction Skills in Higher Education
An Application of the PGK Framework

Zhang, Xiaoling; Aivaloglou, Efthimia; Liut, Michael; Specht, Marcus

DOI
10.1109/EDUCON62633.2025.11016629

Publication date
2025

Document Version
Final published version

Published in
EDUCON 2025 - IEEE Global Engineering Education Conference, Proceedings

Assessment of Algorithmic Abstraction Skills in Higher Education: An Application of the PGK Framework

1th Xiaoling Zhang
EEMCS
Delft University of Technology
Delft, Netherlands
X.Zhang-14@tudelft.nl

2th Efthimia Aivaloglou
EEMCS
Delft University of Technology
Delft, Netherlands
E.Aivaloglou@tudelft.nl

3th Michael Liut
Mathematical and Computational Sciences
University of Toronto Mississauga
Toronto, Canada
michael.liut@utoronto.ca

4th Marcus Specht
EEMCS
Delft University of Technology
Delft, Netherlands
M.M.Specht@tudelft.nl

2025 IEEE Global Engineering Education Conference (EDUCON) | 979-8-3315-3949-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/EDUCON62633.2025.11016629

***Abstract*—Computational Thinking (CT), particularly abstraction, is essential in engineering education, enabling students to break down complex systems into manageable parts. Abstraction helps learners focus on key elements of a problem, ignoring extraneous details. The PGK framework, suggested by Perrenet, Groote, and Kaasenbrood, defines abstraction across four cognitive levels: problem, algorithm, program, and execution. At a higher education institution that focuses on engineering education, we assessed students' abstraction skills using sorting algorithms, chosen for their foundational role and suitability for testing such skills. Our study focused on two areas: (1) the performance of computer science (CS) and non-CS students on algorithmic abstraction tasks, and (2) how factors like demographics, training, programming proficiency, and self-assessed abstraction mastery correlate with task performance. Results showed that all students, especially non-CS majors (including Engineering), need stronger skills at the algorithm, program (coding algorithms), and execution (code functionality) levels. Many non-CS students overestimated their abilities, highlighting a gap in mastery. Students with programming experience performed better, underscoring the importance of hands-on training. These findings suggest interventions for non-CS students are needed to gain experience in programming and to bridge the gap between perceived and actual skills. Future research should focus on discipline-specific curricula and long-term studies to ensure that all students develop the essential CT skills for the digital era.**

***Index Terms*—Computational Thinking, Higher Education, Algorithmic Abstraction Skills, PGK Framework, Assessment**

I. Introduction

In engineering education, algorithmic problem-solving serves as a foundational skill for developing computational solutions and fostering computational thinking (CT) across diverse engineering disciplines. For both Computer Science (CS) and non-CS engineering fields, the ability to approach complex problems through systematic, algorithm-driven methods enhances students' ability to model, analyze, and solve real-world challenges. This not only strengthens their understanding of computational concepts but also equips them with the critical thinking and adaptability needed to innovate in increasingly digital and data-driven industries.

Abstraction is a key element of CT, essential for understanding and solving problems, and translating them into algorithmic solutions [1]. In algorithmic problem-solving, abstraction involves simplifying the problem by removing irrelevant details while keeping its core elements. This process includes refining the solution at different levels until the algorithm is clear and detailed and then converting it into a program. According to Nakar et al. [2], algorithmic problem-solving consists of reformulating the problem, designing an algorithm, implementing it in a programming language, and testing the solution. This method uses a specific form of abstraction known as algorithmic abstraction.

While Nakar et al. [2] believe that such a method can promote CT, courses often cover only part of algorithmic problem-solving [1]. The relevance of different aspects of algorithmic problem-solving for higher education students remains unexplored. While endeavors are made to promote CT in engineering education, including both CS and non-CS contexts, most assessments of abstraction skills and endeavors for CT education promotion in higher education focus on self-evaluation and programming tasks in CS. However, according to [3], CT education should vary regarding the required skills in the corresponding contexts. The PGK framework, illustrated in Perrenet et al.'s work [4], describes four levels of algorithmic abstraction: problem, algorithm, program, and execution levels. This framework has been adopted in various contexts [5]–[8]. In higher education specifically, Aharoni [5] found that undergraduate students struggled with abstract data structure; Perrenet et al. [6] validated a tool for measuring thinking levels with algorithm questions; Brieven and Donnet (2024) taught abstraction in Introduction to Programming

(CS1) course [7]. While this framework has been mainly implemented in CS contexts, the nature of the framework as a thinking tool allows it to be applied in a wider context, such as promoting CT in wider engineering education. The application of the PGK framework to evaluate algorithmic abstraction skills across a broader audience in higher education, including non-CS students, however, is yet to be explored.

In this study, we designed an instrument applying the PGK framework to assess algorithmic abstraction skills for CS and non-CS students in an higher education institution which focuses on engineering education. The algorithmic concept used in the design is sorting, which is a fundamental CS principle and a familiar concept in everyday life. The research questions we aim to answer are:

RQ1 What are the differences in performance on algorithmic abstraction tasks for CS and non-CS students?

RQ2 Which demographic, training, and self-perception factors are associated with performance on algorithmic abstraction tasks?

This paper is organized as follows: Section II presents relevant background and related research results. Section III describes the method used for the study, after which results and analysis will be presented in Section IV. We provide concluding discussions on the results in Section V.

II. Related Work

Studies on CT across different contexts and age levels show that students often struggle with understanding and using abstraction, especially algorithmic abstraction [9]. Methods for teaching algorithmic abstraction vary: some courses use abstraction implicitly without emphasizing the use of it, while others explicitly emphasize the use of it. Findings suggest that explicitly teaching abstraction is more effective for meaningful learning.

Many studies assess abstraction and algorithmic abstraction. For instance, Cutts et al. [9] developed "the abstraction transition taxonomy" to evaluate the degree of abstraction present in tasks, with three levels of abstraction: English, CS speak, and Code. Moreover, Hazzan's theoretical framework [10] on abstraction levels has demonstrated significant relevance for studies addressing abstraction in CS [11], [12]. According to Hazzan [10], students initially perceive new concepts at a lower level of abstraction to reduce cognitive effort. While this may be a temporary learning stage, some students might remain at this lower level, limiting their understanding.

Building on Hazzan's work [10], Perrenet et al. [4] developed the PGK hierarchy to evaluate undergraduate students' understanding of algorithms. The PGK hierarchy has four levels that correspond to different cognitive abstraction stages necessary for a comprehensive understanding of an algorithm. Each higher level is an abstract representation of the one below it, and each lower level is a concrete representation of the one above it. The levels are:

- Level 4: *Problem Level* - This is the highest level, where one can reason about an algorithm in terms of the problem it solves and its characteristics. At this level, an algorithm is perceived as a black box, requiring a high degree of abstraction.
- Level 3: *Algorithm (Object) Level* - Here, an algorithm is seen as an object independent of any specific programming language. Complexity measures such as time and space efficiency become relevant at this stage.
- Level 2: *Program (Process) Level* - At this level, an algorithm is viewed as a process, described by a specific executable program written in a particular programming language.
- Level 1: *Execution Level* - The lowest level, where an algorithm is interpreted as a specific run on specific input on a concrete machine.

Following Perrenet et al. [4], many studies have used the PGK hierarchy to examine K-12 students' understanding of algorithmic abstraction [1], [11], [13], [14]. For example, it has been applied in primary school curricula, especially with teachers, where the *Algorithm Level* is seen as a preliminary design phase [1]. In higher education, Aharoni used the PGK hierarchy to study undergraduate students in a data structure course and found that they generally understand data structures at the *Algorithm Level* [5]. Brieven and Donnet (2024) [7] taught abstraction in Introduction to Programming (CS1) course and their findings emphasized the importance of illustrating to students the function of abstraction in problem-solving and its relevance to work life.

III. Method

A. Study Design

This cross-sectional study followed Bethlehem's procedure [15]. We designed the material to collect participants' demographic information, training experience in programming, CS, and algorithmic problem solving, self-perception in programming language proficiency, self-perception on mastery and necessity of algorithmic abstraction skills, and performance on algorithmic abstraction tasks. Items were revised iteratively based on expert feedback and trials with participants. Data were analyzed using basic descriptive statistics and qualitative analysis.

B. Participants - Population and Sampling

The study involved Bachelor, Master, and PhD students across diverse disciplines in higher education in the Netherlands. To comply with GDPR and facilitate practical survey research, we applied non-probabilistic sampling methods like convenience and referral sampling. Approval was obtained from the Human Research Ethics Committee of Delft University of Technology. Convenience sampling targeted easily accessible participants through QR codes, online and physical leaflets, social media, and the university's recruitment system. Snowball sampling involved participants inviting others to join the study. Before participating, individuals received an informed consent form detailing the research's purpose. A total of 30 respondents completed the questionnaire and received a 10 Euro compensation voucher upon survey completion.

C. Materials - Instrumentation

The experiment material[1] included questions on: (1) participants' demographic information, training experience in programming, CS, and algorithmic problem solving, self-perception on programming languages proficiency; (2) assessment of participants' algorithmic abstraction skills in solving everyday problems; and (3) self-perception on their mastery and the importance of algorithmic abstraction skills.

To begin with, for (1), participants were requested to provide demographic details, including gender (Q1), age range (Q2), level of the study program (Q3), year of study (Q4), study domain (Q5-Q6), past training experience in Computer Science, programming, Computational Thinking, or algorithmic problem-solving (Q7), as well as proficiency with programming languages (Q8). The questions asked in this part are mostly multiple-choice questions and Likert scales.

For (2), participants completed tasks based on the PGK hierarchy. Tasks at the *Problem Level* (SP1-Q1 to SP1-Q3) tested their ability to recognize algorithms by the problems they solve, define algorithms, and apply them in different scenarios. Moving to the *Algorithm Level* (SP2-Q1 to SP2-Q6), tasks evaluated participants' skills in creating step-by-step procedures for solutions, considering various versions of procedures and the complexity of the solution, and expressing them in pseudo-code or natural language. At the *Program Level* (SP3-Q1 to SP3-Q2), participants rearranged disorderly code implementations of algorithms and implemented solutions using Python. Finally, tasks at the *Execution Level* (SP4-Q1 to SP4-Q3) focused on analyzing implementations' execution, debugging issues, and providing strategies for verifying implementation functionality. In this part, almost all questions are open-text questions, a few are multiple-choice and re-ordering questions.

For (3), in two multiple-choice questions (from P-Q1 to P-Q2), the participant was asked to reflect on the necessity of the level of algorithmic abstraction skills and their perceived mastery level. This part only contains multiple-choice questions.

D. Data collection

We created an online environment on the Qualtrics platform with the described multiple-choice and rating questions, along with the tasks covering the four levels of algorithmic abstraction as described earlier. Participants were assigned anonymous IDs unrelated to their backgrounds to ensure confidentiality. Before starting, participants received guidance on using the platform and seeking assistance. During the survey, a researcher remained nearby to offer support as needed. Each session lasted approximately one hour.

The experiment yielded 30 records in the dataset. After reviewing the dataset, all valid data underwent pre-processing for further analysis. This process involves data cleaning, transformation, coding, and aggregation.

[1]Available at https://docs.google.com/document/d/1JwuM7K8O2GJw2KlZbTzAAbqLYCx2CPLP0ZDQtxPdNng/edit?usp=sharing

E. Data Analysis

To understand participants' demographics, training experience in CS or algorithmic problem-solving, programming language proficiency, and perceptions on mastery and necessity of algorithmic abstraction skills, we first applied descriptive analysis to questions Q1 to Q8. These questions are in multiple-choice and Likert scale formats.

To answer RQ1, we scored participants' algorithmic abstraction skills using a rubric[2] based on the PGK framework. Since each level had different total scores, we used Min-Max normalization to standardize the data, transforming the minimum value to 0, the maximum to 1, and all other values to decimals between 0 and 1. We then used an independent student's t-test on the normalized scores to check for significant differences between CS and non-CS participants.

Since this study applies an existing theretical framework on algorithmic abstraction, we analyzed tasks at different levels of algorithmic abstraction using deductive thematic analysis based on Kiger et al.'s [16] work to guide the coding and theme development. Initial codes were generated from the rubric criteria used for scoring tasks. These codes were then organized into themes for each level of algorithmic abstraction as defined by the PGK framework. For the *Problem Level*, we analyzed SP1-Q2 and SP1-Q3, where participants defined the concept of sorting and gave examples of its daily or professional applications.

For the *Algorithm Level*, we analyzed SP2-Q3, SP2-Q2, and SP2-Q4. Participants reasoned the applicability of an algorithm, considered its complexity and provided a step-by-step plan. For the *Program Level*, we analyzed SP3-Q2, where participants implemented their proposed algorithm and reflected on implementation problems. For the *Execution Level*, we analyzed SP4-Q3, where participants described their debugging strategies for a piece of buggy code. The final hierarchy is shown in Table I.

To answer RQ2, we first performed a basic statistical analysis of the multiple-choice answers (P-Q1 and P-Q2) to study their perceptions of the mastery and necessity of algorithmic abstraction skills. We then conducted a correlational analysis to examine how demographic factors (Q1-Q6), programming language proficiency and training experience in CS or algorithmic problem-solving (Q7-Q8), and perceptions of the mastery and necessity of algorithmic abstraction skills (P-Q1 and P-Q2) are associated with participants' scores on algorithmic abstraction tasks.

IV. Results and Analysis

Table II shows the analysis of results from Q1, Q2, Q3, Q6, and Q7, results of Q4 and Q5 were omitted in the table for ethical data reporting since some data points can be easily singled out. Most participants are males aged 21 to 30 and enrolled in Master of Science programs. All participants are from STEM

[2]The rubric can be accessed via author requests.

[3]Training experience in Computer Science, Programming, CT, or Algorithmic Problem-Solving.

TABLE I
THEMATIC ANALYSIS HIERARCHY FOR QUALITATIVE DATA FROM EACH LEVEL OF ALGORITHMIC ABSTRACTION TASKS

Dimensions	Themes	Codes
Problem Level	Different understanding of an algorithm concept *(Answers from SP1-Q2 and SP1-Q3)*	Sorting as grouping definition as well as the application of sorting in daily and professional life.
		Sorting as ordering definition as well as the application of sorting in daily and professional life.
		Sorting as both grouping and ordering.
Algorithm Level	Different levels of consideration over the complexity of the algorithm *(Answers from SP2-Q3)*	Consideration over the complexity of the problem.
		Consideration of the strategy for solving the problem.
	Levels of concreteness on the step-by-step plan for the solution *(Answers from SP2-Q2 and SP2-Q4)*	Generic plan on a strategy that does not contain specific operational steps.
		Concrete plan that includes the basic operation of the algorithm.
Program Level	Problems with Implementation *(Answers from SP3-Q2)*	Time, Indication of using existing functions, Python proficiency
Execution Level	Level of specificity on debugging strategies *Answers from SP4-Q3*	Detailed plan for debugging and execution.
		Generic descriptions of overall strategy.

TABLE II
DESCRIPTIVE ANALYSIS OF THE SAMPLE (Q1-Q7)

Items	Category	Frequency	%
Gender	Female	17	56.7 %
	Male	11	36.7 %
	Others	2	6.6 %
Age	20 or below 20	2	6.7 %
	21-30	25	83.3 %
	31 and above	3	10 %
Education Level	Bachelor of Science (BSc)	6	20.0 %
	Master of Science (MSc)	18	60.0 %
	Ph.D.	6	20.0 %
Study Subject	CS	12	40.0 %
	Non-CS	18	60.0 %
Experience[3]	Never	1	3.3 %
	Once	3	10.0 %
	Sometimes	12	40.0 %
	Repeatedly	4	13.3 %
	Regularly	10	33.3 %

fields, with the majority having a non-CS background. Eight of the non-CS participants come from the applied sciences or technology fields. This results in 22 engineering participants (CS participants and other engineering participants) and eight non-engineering participants. Additionally, nearly all participants have had more than one training experience in CS, CT, programming, or algorithmic problem-solving. Analysis of results from Q8 indicates that all participants reported low competence in at least one programming language. Python was the most commonly cited language in which participants had above-average competence.

TABLE III
DESCRIPTIVE STATISTICS FOR DIFFERENT ALGORITHMIC ABSTRACTION LEVELS AND TOTAL SCORES

Levels	Domain	N	Mean	SD[4]
Problem Level	CS	12	0.840	0.1061
	Non-CS	18	0.714	0.1511
Algorithm Level	CS	12	0.449	0.1598
	Non-CS	18	0.389	0.1349
Program Level	CS	12	0.389	0.1436
	Non-CS	18	0.366	0.1489
Execution Level	CS	12	0.286	0.1518
	Non-CS	18	0.257	0.1544
All Levels	CS	12	0.471	0.0902
	Non-CS	18	0.412	0.0861

TABLE IV
NORMALITY TEST AND STUDENT'S T-TEST OF SCORES FOR DIFFERENT ALGORITHMIC ABSTRACTION LEVELS AND TOTAL SCORES

Levels	Shapiro-Wilk normality test (p-value) [5]	p-value [6]
Problem Level	0.025	**0.009**
Algorithm Level	**0.098**	0.139
Program Level	<0.001	0.338
Execution Level	0.019	0.305
All Levels	0.356	**0.041**

A. RQ1 The Differences in performance on algorithmic abstraction tasks for students with CS or non-CS backgrounds

For this research question, we analyzed and run the Shapiro-Wilk normality test (appropriate for small sample size that is less than 50) on the scores of tasks, with an independent Student's t-test being an additional measurement.

1) Analysis on the scores: To examine whether the CS background plays an important role in the level of algorithmic abstraction skills, we started with a descriptive analysis of scores. Table III shows that participants with a CS background obtained a higher average score for every level of algorithmic abstraction skills. Furthermore, the results show a trend of a decrease in the average score from the *Problem Level* to the *Execution Level*, and the participants obtain more than half of the scores on the *Problem Level*.

Furthermore, we conducted a normality test and an independent samples T-test to examine if the scores of the CS participants were normally distributed and significantly higher than those of the non-CS participants, respectively. Table IV show that only the scores of the *Algorithm Level* conforms a normal distribution. Due to the small sample size of the study, we proceed with the T-test analysis. Table IV shows that there is a significant difference between the CS and the non-CS groups regarding the scores for the *Problem Level* and

[4]SD stands for standard deviation.

[5]Shapiro-Wilk normality test p-value, yhe test rejects the hypothesis of normality when the p-value is less than or equal to 0.05

[6]Student's T-test for scores at different levels of algorithmic abstraction skills with CS or Non-CS being the grouping variable.

the sum of all levels, but not a significant difference regarding the *Algorithm Level*, *Program Level*, and the *Execution Level*.

2) Thematic analysis: At the *Problem Level*, participants demonstrated various understandings of the concept of an algorithm. Definitions of sorting varied, with participants applying it to different scenarios. Four non-CS participants defined sorting as a technique for grouping items. The remaining participants described sorting as a method for ordering items or as a technique that can be used for both grouping and ordering. Here are examples of each category from their answers. **Sorting as a technique to group things:** *"Sorting is dividing a larger group of items into different smaller groups, where they have a distinct feature.", "I work in administration for my football association, and I work a lot with excel where for each member there are some yes-no questions, for example, if they have completed some steps in the registration process. Then it is very easy for me to sort the whole list, by selecting the members that have not yet sent me a certain document, and I will contact them."* **Sorting as a technique to put things into a certain order:** *"Sorting is ordering items based on properties that can be compared to each other and said to be higher, lower, or equal.", "When creating backups of my files, going from smaller to bigger files ensures the biggest amount of files is uploaded, in case the internet cuts off halfway through."* **Sorting as a technique for both grouping things and putting things into a certain order:** *"Sorting is the act of arranging a set of elements in order or in groups based on some characteristic.", "I sort my clothes based on when I last wore them, so that I go through my whole closet, and do not repeat the same clothes too soon."*

At the *Algorithm Level*, we examined participants' approaches to algorithm design and the concreteness of their step-by-step plans. Seven CS-background participants considered the complexity of the problem, while the rest focused on problem-solving strategies or less relevant aspects for computational efficiency. Regarding the step-by-step plans, 11 participants (including four from a CS background) provided generic, strategic plans lacking specific operational steps. Most participants, however, offered concrete plans detailing basic algorithm operations such as *compare* and *swap*. **Consideration of the complexity of the problem:** *"The way to sort a deck of cards can be used to organize over 5,000 books, but I think it might be computationally intensive. especially as letters are involved and strings need to be compared ..."* **Consideration of the strategy for solving the problem or other aspects:***"The way to sort a deck of cards cannot be used to organize over 5,000 books, it depends on which properties you divide them."* **Generic plan on a strategic level that does not contain specific operational steps:***"In the desired order the cards are arranged numerically in increasing order. So I would just look for the card with a small number and keep it to the left."* **Concrete plan that includes the basic operation of the algorithm:** *"Start on the left side. Pick the first (leftmost) card. Compare it to the card to the right of it. If the card is higher than its rightside neighbour, swap them. Do this until you don't swap anything anymore and move on to the next card."*

At the *Program Level*, we analyzed the reflections of seven participants who did not implement the algorithm in Python. Their reflections were categorized into three main areas: time constraints, Python proficiency, and reliance on existing functions. Here are examples for each category. **Time:** *"It takes time for me to come up with a logic step by step and implement it."* **Python Proficiency:** *"I am not able to generate Python code spontaneously."* **Indication of using existing functions:** *"I would normally not sort anything myself since sorting algorithms are already implemented and available in most languages."*

At the *Execution Level*, eight participants, including five with a non-CS background, provided concrete plans for debugging and execution. The remaining participants gave more generic descriptions of their overall debugging and execution strategies.**Detailed plan for debugging and execution:** *"Write out the effects of the code in an example calculation, use a supporting code editor error analyzer", "Check syntax errors, debugging print statements, stepwise execution, logically thinking about the algorithm the code is following."* **Generic descriptions of overall strategy:** *"Run with test example" or "Write some tests with simple examples."*

B. RQ2 Factors that are associated with one's performance on algorithmic abstraction tasks

To begin with, we conducted a basic statistical analysis of the answers on perceptions of mastery and the necessity of algorithmic abstraction skills. Table VI shows that the variance is larger in the rating of the mastered level, with the CS participants achieving a higher mean and median. This indicates that the perceived level of skill mastered for CS participants is on average higher than that of non-CS participants. Moreover, the results show that the lowest level needed of algorithmic abstraction skills is at the *Algorithm Level* for both CS and non-CS participants, and most of the participants thought it needed to master the *Program Level* or even the *Execution Level*. Meanwhile, for the perceived needed level of algorithmic abstraction skills, both the mean and median level of algorithmic abstraction skills for CS participants is close to that of non-CS participants. However, the gap between the mastered level and the needed level is higher for non-CS participants than for CS participants.

Regarding how factors correlate with student performance, the results show no significant correlation between the total score across all tasks and demographic factors. Table V illustrates that these total scores correlate significantly with the training experience of the participants and the programming language proficiency of the participants. Meanwhile, there is no significant correlation between total scores and perception of the level mastered and the level needed for algorithmic abstraction skills, respectively. Furthermore, the analysis shows that training experience correlates significantly with programming experience, and study subjects correlate significantly with both training and programming experience.

TABLE V
CORRELATIONAL ANALYSIS BETWEEN THE TOTAL SCORE AND DEMOGRAPHIC VARIABLES

		Gender	Age	Education Level	Study Subject	Training Experience	Programming language proficiency	Perceived Mastered Level	Perceived Needed Level
Score_All Levels	Person's r	0.143	-0.114	0.132	-0.316	0.402*	0.423*	0.223	-0.112
	p-value	0.450	0.548	0.487	0.089	0.028	0.020	0.237	0.556

Note: * $p < 0.05$

TABLE VI
PARTICIPANTS' PERCEPTIONS OF THE MASTERED AND LEVEL OF ALGORITHMIC ABSTRACTION SKILLS

	Mastered Level		**Needed Level**	
	CS	Non-CS	CS	Non-CS
Mean	3.58	2.44	3.50	3.17
Median	4	2	4	4
Standard Deviation	0.793	1.04	0.905	0.985
Minimum	2	1	2	2
Maximum	4	4	4	4

V. DISCUSSION AND CONCLUSION

A. Limitations of this study

The study has several limitations. Firstly, the findings are based on a small sample size from a specific context in the Netherlands, and the sample does not conform to a normal distribution, limiting their generalizabilit. To enhance generalizability, replication in diverse contexts and a larger sample size is crucial. Secondly, the sampling methods used are constrained by practical limitations, which also affect the study's broader applicability. Moreover, maintaining rigor in the thematic analysis of qualitative data is challenging. Although the researcher aimed to assist participants only upon request for technical issues or clarification on the process or questions, their presence during data collection could still influence participant responses and the interpretation of results.

B. Implications on education on algorithmic abstraction skills

Based on the findings from RQ2, both CS and non-CS participants indicated the lowest perceived need for algorithmic abstraction skills at the *Algorithm Level*. Moreover, most participants expressed a necessity to master skills at the *Program Level* or the *Execution Level*. Since nearly all participants are from STEM fields, we recognized the need to cultivate STEM students with at least algorithmic abstraction skills at the *Algorithm Level*, and potentially up to *Execution Level* depending on their backgrounds. Furthermore, non-CS participants show a larger gap between their perceived mastery and their perceived need than CS participants. Addressing these discrepancies is crucial for designing educational strategies that meet students' needs, requiring carefully examining challenges and potential solutions.

The performance differences in algorithmic abstraction tasks between CS and non-CS participants highlight significant educational challenges. CS participants consistently scored higher across all levels compared to non-CS participants, with statistically significant differences observed mainly at the problem level and in the total scores across all levels. This suggests that CS participants may encounter fewer obstacles in understanding and reformulating algorithmic problems. This finding correlates with the observation that non-CS participants tend to define algorithm concepts differently, often diverging from computational contexts. In contrast to Aharoni's findings [5], where understanding was centered at the *Algorithm Level* of data structures, participants in this study typically grasp algorithmic abstraction at the *Problem Level*. Potential contributing factors include differences in the application context of the PGK framework, variations in the assessment instruments derived from the framework, and differences in sample composition compared to Aharoni's study [5].

There are also notable performance differences observed at other levels of algorithmic abstraction skills. At the *Algorithm Level*, CS participants demonstrate greater awareness of computational complexity in algorithm design compared to non-CS participants. This finding is consistent with a study [17] that explored the integration of CT in engineering curricula, highlighting that non-CS students have less emphasis on mastering computational design compared to CS students. At the *Program Level*, concerns revolve mainly around time constraints and proficiency in programming languages when implementing algorithms. In addition, only a minority of participants offered detailed plans for the debugging and execution of the algorithms.

The results of this study show that previous experience is a strong predictor of performance on the assessment instrument, which supports the validity of the instrument. Although Table IV indicates a significant association between the study subject and the scores, Table V reveals that the study subject is not a predictive factor of overall performance when factors such as training experience and language proficiency are taken into account. This suggests that experience, rather than study subject, mediates performance outcomes. However, since the tasks are designed with a CS profession's mindset, the interpretation of the results can be biased. For example, the answers to the tasks on the *Algorithm Level*, the results may not necessarily imply that non-CS participants have weaker computational complexity awareness. Rather, it may suggest that they approach or contextualize these tasks differently. Furthermore, even if non-CS students exhibit lower awareness, the results suggest that experience, rather than discipline, is the primary predictor of performance. The data imply that non-CS students can achieve similar levels of proficiency as CS students with additional experience, suggesting that experience-focused interventions may be more beneficial than

discipline-specific ones.

The findings from RQ2 suggest potential strategies for addressing challenges identified in the study. Building on these findings and considering the performance disparities revealed in RQ1, with factors influencing performance identified in RQ2, it is recommended to prioritize training in CS, CT, programming, and algorithmic abstraction skills to enhance students' abilities in algorithmic abstraction. Furthermore, tailored training programs should address specific needs across different backgrounds. For STEM students in general, achieving proficiency at the *Algorithm Level* appears crucial. This can be facilitated through discussions on algorithmic concepts and introducing computational complexity, which is particularly beneficial for non-CS students. Educational initiatives aiming to enhance skills at the *Program Level* should account for learners' familiarity with programming languages and the time required for implementation. The challenges and impact of detailed debugging and execution plans across different domains warrant further investigation, particularly at the *Execution Level*.

C. Implications for future research

This research operationalized the PGK framework to investigate differences in student performance on algorithmic abstraction tasks, and how demographic factors, training experience, programming proficiency, and students' perceived and actual mastery of these skills are correlated to their performance. As CT gains more attention among non-CS students [18] we offer several suggestions for future work based on the findings of this study. According to our results future research should focus on several key areas: first, to develop domain-specific curriculum modules to address gaps in understanding algorithm concepts and computational complexity; second, designing comprehensive training programs in CT and programming for both CS and non-CS students to bridge the gap between needed and actual skill levels; third, conducting longitudinal studies to monitor the development of algorithmic abstraction skills and the effects of different educational interventions over time; fourth, fostering cross-disciplinary collaboration to integrate algorithmic thinking into non-CS curricula; and fifth researching effective methods for teaching computational complexity to non-CS students, evaluating the benefits of introducing these concepts early in the curriculum.

Moreover, considering the differences found between CS and non-CS students, which show that the training experience and the programming experience are vital in enhancing CT education; the authors would suggest to further improve CT education in engineering curricula. Specifically, future studies are suggested to replicate this study with students from non-CS engineering and non-CS non-engineering backgrounds. This could provide insights into the performance differences between the non-CS engineering and non-engineering participants.

REFERENCES

[1] M. Armoni, "On teaching abstraction in cs to novices," *Journal of Computers in Mathematics and Science Teaching*, vol. 32, no. 3, pp. 265–284, 2013.

[2] L. Nakar, M. Friebroon, and M. Armoni, "From modelling to assessing algorithmic abstraction–the missing dimension," in *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*, 2023, pp. 1–12.

[3] P. J. Denning and M. Tedre, *Computational thinking*, ser. The MIT press essential knowledge series. Cambridge, Massachusetts: The MIT Press, 2019.

[4] J. Perrenet, J. F. Groote, and E. Kaasenbrood, "Exploring students' understanding of the concept of algorithm: levels of abstraction," *ACM SIGCSE Bulletin*, vol. 37, no. 3, pp. 64–68, 2005.

[5] D. Aharoni, "Cogito, ergo sum! cognitive processes of students dealing with data structures," in *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, 2000, pp. 26–30.

[6] J. Perrenet and E. Kaasenbrood, "Levels of abstraction in students' understanding of the concept of algorithm: the qualitative perspective," *ACM SIGCSE Bulletin*, vol. 38, no. 3, pp. 270–274, 2006.

[7] G. Brieven and B. Donnet, "Practicing abstraction through a top-down problem-solving framework in a cs1 course," in *10th International Conference on Higher Education Advances (HEAd24)*. Universitat Polytecnica de Valencia, Valence, Unknown/unspecified, 2024.

[8] I. L. Graafsma, S. Robidoux, L. Nickels, M. Roberts, V. Polito, J. D. Zhu, and E. Marinus, "The cognition of programming: logical reasoning, algebra and vocabulary skills predict programming performance following an introductory computing course," *Journal of Cognitive Psychology*, vol. 35, no. 3, pp. 364–381, 2023.

[9] Q. Cutts, S. Esper, M. Fecho, S. R. Foster, and B. Simon, "The abstraction transition taxonomy: Developing desired learning outcomes through the lens of situated cognition," in *Proceedings of the ninth annual international conference on International computing education research*, 2012, pp. 63–70.

[10] O. Hazzan, "Reducing abstraction level when learning abstract algebra concepts," *Educational Studies in Mathematics*, vol. 40, pp. 71–90, 1999.

[11] M. Friebroon-Yesharim, R. Ben-Bassat Levy, and M. Armoni, "Algorithmic abstraction in computer science curricula for primary schools: The case of a national curriculum for 4th grade," in *Proceedings of the 2023 Conference on United Kingdom & Ireland Computing Education Research*, 2023, pp. 1–7.

[12] J. Hartmanis, "Turing award lecture on computational complexity and the nature of computer science," *Communications of the ACM*, vol. 37, no. 10, pp. 37–43, 1994.

[13] D. Statter and M. Armoni, "Teaching abstraction in computer science to 7th grade students," *ACM Transactions on Computing Education (TOCE)*, vol. 20, no. 1, pp. 1–37, 2020.

[14] L. Nakar and M. Armoni, "On teaching abstraction in computer science: Secondary-school teachers' perceptions vs. practices," in *Proceedings of the 2023 Conference on United Kingdom & Ireland Computing Education Research*, 2023, pp. 1–7.

[15] J. Bethlehem, *Applied survey methods: A statistical perspective*. John Wiley & Sons, 2009.

[16] M. E. Kiger and L. Varpio, "Thematic analysis of qualitative data: Amee guide no. 131," *Medical teacher*, vol. 42, no. 8, pp. 846–854, 2020.

[17] X. Zhang, M. Specht, and M. Valle Torre, "An investigation on integration of computational thinking into engineering curriculum at delft university of technology," in *Towards a new future in engineering education, new scenarios that european alliances of tech universities open up*. Universitat Politècnica de Catalunya, 2022, pp. 890–901.

[18] X. Zhang, F. Aivaloglou, and M. Specht, "A systematic umbrella review on computational thinking assessment in higher education." *European Journal of STEM Education*, vol. 9, no. 1, p. 2, 2024.