Integrating Computational Thinking Into the Curricula to Bridge the Skill Gap in Engineering Education

Nasrin Dehbozorgi
Dept. of Software Engineering
Kennesaw State University
Marietta, GA, USA
dnasrin@kennesaw.edu

Maysam Nezafati
Dept. of Biomedical Engineering
Georgia Institute of Technology
Atlanta, GA, USA
maysam.nezafati@bme.gatech.edu

Mehdi Roopaei
Dept. of Electrical and Computer Engineering)
University of Wisconsin-Platteville
Platteville,WI, USA
roopaeim@uwplatt.edu

2024 IEEE Frontiers in Education Conference (FIE) | 979-8-3503-5150-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/FIE61694.2024.10892914

***Abstract*—This work-in-progress research-to-practice paper presents an intervention on integrating computational thinking modules into a software engineering course. The national consensus on the significance of computational thinking has prompted the expansion of related educational initiatives over the past decade. Since the definition of computational thinking by Wing in 2006, this concept has gained significant attention within the educational community. Particularly this surge of interest has led to extensive research into its conceptual foundations and subsequent integration into educational curricula since 2013. National initiatives have since emerged to incorporate computational thinking into the educational system. Furthermore, as artificial intelligence and computing systems become increasingly integrated into daily life, there is a growing demand from industries for a workforce and graduates adept at critical thinking and problem-solving. Aligned with this national movement, our study presents a two-year institutional initiative, aimed at integrating computational thinking into the software engineering program. The software engineering discipline extensively involves design thinking and problem-solving skills. However, we noticed that these higher-level skills are not imparted early in the program to teach students this method of thinking and approaching problems. To bridge this skill gap, we developed a set of computational thinking modules and integrated them into a gateway course in the software engineering program. Over two years, we implemented this intervention in an introductory-level course and evaluated its impact on students' computational thinking skills by analyzing their responses to a standard Computational Thinking Assessment survey. The results showed significant improvement in most components. These early findings underscore the effectiveness of integrating these computational thinking modules into the gateway courses, regardless of the specific course topic. A notable feature of these modules is their adaptability to diverse engineering courses, suggesting broader applicability across disciplines. Moving forward, our research aims to expand the integration of the computational thinking modules into various courses in other institutes across the nation and analyze their impact on student performance.**

***Index Terms*—Computational Thinking, Software Engineering, Engineering Education**

I. Introduction

In today's AI-driven era, where technology permeates every aspect of our lives, Computational Thinking (CT) rises as a foundational skill for students, especially those in STEM fields. CT cultivates the mindset of a computer scientist in students by emphasizing critical concepts like decomposition, pattern recognition, algorithmic design, abstraction, data representation, simulation, and evaluation [1]. By imparting these foundational skills, the educational system prepares graduates empowered not only to comprehend but also to influence the technologies that shape our future. With the growing recognition of CT in educational and professional domains, educators and policymakers acknowledge the pressing need to integrate it into educational curricula in recent years [2]. However the initiatives to improve CT skills among students in higher education encounter different opportunities and challenges, particularly concerning their practical implementation and evaluation methodologies across diverse disciplines. Historically, CT was conceptualized as the systematic approach to problem-solving using computational tools. However, contemporary perspectives define CT as a fundamental cognitive framework that fuses aspects of mathematical and engineering thinking. Furthermore, recent research characterizes it as the creative process of devising innovative solutions to complex problems [2], [3]. Although traditionally CT was rooted in computer science, it has transcended disciplinary boundaries, and has emerged as a skill similar to creativity. This evolution has expanded the research focus beyond CS to other domains within STEM fields [4]. However, despite this expansion, there remains a gap in integrating CT into the engineering curricula which underscores the need for further efforts to incorporate it into these fields [4].

In line with the national movement towards integrating CT into education, our study introduces a two-year institutional initiative aimed at incorporating CT principles into the software engineering program [5]. Despite its emphasis on coding and programming, the software engineering discipline demands a robust foundation in design thinking and problem-solving skills. Tasks such as architecting complex systems, developing subsystems, and customizing solutions necessitate a high level of CT competency, which includes abstraction, decomposition, pattern recognition, and algorithm design (Figure 1).

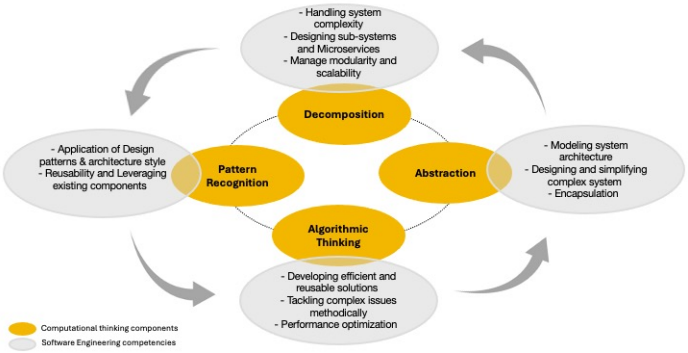

Fig. 1. Mapping of SWE competencies with CT skills

However, our observation revealed a gap in the early cultivation of these higher-level skills within the program, which hindered students' ability to approach problems with a CT mindset. To address this issue, we developed a set of CT modules informed by existing research and CT best practices, and integrated them into a gateway course. These modules were designed to impart problem-solving tactics and strategies without requiring prior programming knowledge from students. The early result of this initiative shows promising by improving students' creativity and problem solving skills. In the next section, we overview prior research on integrating CT into curricula. Then, we detail our methodology and findings on the impact of our intervention. Finally, we discuss our results and outline future work plans.

II. Related Work

The importance of CT is increasingly recognized in STEM education. It's considered crucial for maintaining national competitiveness in the workforce. Given this emphasis, it's important for the educational community to explore effective methods for integrating CT skills across all educational levels in various majors and disciplines [2], [6]. Despite the critical need for developing CT skills, there is limited practical focus on this aspect in engineering education. Research suggests that one of the significant challenges in integration of the CT into educational programs is rooted in its ambiguous definition in the literature. This ambiguity stems from diverse definitions and frameworks in the literature [2], [6], [7]. We identified that common themes in defining CT include abstraction, algorithmic thinking, decomposition, and pattern recognition. Another challenge relates to pedagogical design and how these competencies can be fostered through class activities and curricula. Furthermore, there is a need to design discipline-specific practices to improve students' CT skills in engineering majors. The existing literature suggests that CT has been mainly served as a framework for designing instruction, activities, content, and assessments in STEM fields. Mainly there are three categorical themes emerged from the studies regarding the application of CT. Some studies structured their courses around CT, others focused on teaching CT skills, while others primarily leveraged CT to improve students' learning in the subject [2]. One of the most common practices of CT are simulation activities which are widely employed in engineering education. Studies emphasize that a fundamental aspect of CT in undergraduate engineering education entails participation in modeling and simulation activities [8], [9], [10]. It is also mentioned by some researchers that programming is the 'ideal' medium for CT implementation [12]. In one study [13], the authors investigated the impact of a programming course on the development of CT in university students. They proposed a programming CT framework that encompassed computational concepts, practices, and perspectives, which they evaluated through an observational study. The study suggests that training is necessary to acquire CT skills that cannot be developed naturally, and programming enhances the acquisition of these skills. Review on the strategies for implementing CT reveals that they mainly involve block-based, game-based, and constructivist methods for programming. In another systematic review, Friday et al. [14] examined the various approaches to implementing CT in higher education and highlight its potential to transform programming education. The review identified five pedagogical methods for CT implementation: unplugged activities, arrow-based visual environments, block-based visual environments, textual programming languages, and CT's integration with the physical world. Their research reports that educators primarily utilized tools such as CT-tests, Bebras, and Dr. Scratch to facilitate CT development among students. However other studies claim while tools such as Scratch can positively affected CT skills, the use of such tools can be 'potentially distracting' for students in gaining general problem-solving skills [15]. The researchers also confirm that some of the challenges are posed by the lack of consensus on CT assessment and standardized measuring instruments [16]. Despite efforts to develop taxonomies for assessing students based on various CT definitions, definitions alone are inadequate without practical pedagogical and assessment methods. In summary, the literature review highlights challenges that hinder the widespread adoption of CT, such as vague definitions, insufficient assessment mechanisms, and unclear classroom applications. Future research should address these challenges by focusing on operational definitions of CT in broader disciplinary contexts, exploring ways of integrating CT into new pedagogies, and examining specific implementations in undergraduate discipline-specific settings. Practical methodologies and design-based research offer promising avenues for developing embedded design principles within disciplinary contexts [2].

III. Methodology

In this study, we propose leveraging CT to enhance computational skills among SWE students, to improve their problem solving skills and performance in their coursework and discipline-based studies. Our intervention introduces a set of unplugged activities that do not require specific computational knowledge or programming skills. These modules revolve around the four common components of CT identi-

fied as abstraction, algorithmic thinking, decomposition, and pattern recognition. Abstraction involves simplifying complex problems, while algorithmic thinking emphasizing step-by-step problem-solving procedures. Decomposition breaks down complex problems into smaller, manageable parts, while pattern recognition involves identifying recurring trends.

We designed a set of four CT modules to teach students how to approach real-world problems. In the Abstraction module, the emphasis was on learning to disregard unnecessary details and concentrate on identifying the key information essential for solving a particular problem. Our observation in upper-level SWE courses revealed that although abstract thinking is a necessary skill for designing and modeling system architecture, students often struggle with it. The second module (Decomposition) teaches students how to decompose problems into smaller manageable parts, a vital skill for SWE in designing subsystems, microservices, and generally developing any system. The third module focused on Pattern Recognition. Its goal was to help students learn how to find common trends, practices, and recurring problems and solutions. This is a key skill for SWE developers to utilize design patterns and architectural styles in designing new systems and leveraging existing best practices and components. Lastly, the Algorithmic Thinking module aims to improve students' algorithmic thinking. This skill is somewhat inclusive of other CT skills, requiring students to break down a problem and create sequential steps to solve it. Each module includes instructional materials, practice exercises, assignments and quizzes to reinforce students' learning. The developed CT modules were integrated into a foundational computing course aimed at introducing students to computing literacy, regardless of their background and prior knowledge in the field. Each module was structured to introduce students to a specific competency through readings, videos, and individual or team assignments. We employed the Holistic Assessment of CT (Hi-ACT) test [11] as a standard measure to evaluate students' CT skill both before and after the intervention. For this study, we tailored a customized version of the Hi-ACT test comprising seven categories: creative thinking, algorithmic thinking, critical thinking, problem-solving, confidence, persistence, and cooperation. The results of the data analysis are elaborated in the subsequent section.

A. Data Analysis

Data for this project were collected from three foundational courses of Computing: Introduction to Computing (A), Programming and Problem Solving I (B), Programming and Problem Solving II (C). These courses are structured such that students first take Course A, followed by Courses B and C in the next semester. We integrated CT modules into Course A and conducted a comprehensive data analysis to determine if students carried these competencies into the upper-level courses (Courses B and C). The intervention was applied starting in Spring 2022. In this study, we report on the pre- and post-intervention survey data of 457 students and compare the results to determine if there is a statistically significant improvement in students' CT skills after the intervention. Surveys were administered to students in these courses during the Fall 2021, Spring 2022, and Fall 2022 semesters. The CT modules were applied in the Spring and Fall of 2022. We received a total of 215 responses in Fall 2021, followed by 139 responses in Spring 2022, and 103 responses in Fall 2022. The survey chosen for this project encompasses seven main categories of Creative Thinking, Algorithmic Thinking, Critical Thinking, Problem Solving, Confidence, Persistence, and Cooperation. Each category consists of several questions in the form of a 5-point Likert Scale (ranging from 'Strongly Disagree' to 'Strongly Agree'), culminating in a total of 50 questions that comprehensively cover all aspects of each CT components. For evaluation purposes, the dataset was transformed, where responses such as 'strongly agree,' 'agree,' 'neutral,' 'disagree,' and 'strongly disagree' were converted to a numeric scale ranging from 5 to 1, respectively. This transformation facilitates the quantitative analysis of the data, making it suitable for statistical evaluation. To analyze the data, the survey results from all students in Fall 2021 were compared separately to those from Fall 2022 and Spring 2022. Individual question responses and cumulative responses within each category were analyzed using two tailed t-tests (alpha = 0.05) to determine significant differences.

1) Results Interpretation: The data analysis results indicate that out of seven categories, there was a statistically significant improvement in CT skills post intervention in four categories of Creative Thinking, Problem Solving, Confidence and Persistence. There was no significant change in two categories of Algorithmic thinking and Cooperation indicating that the CT modules might not have effectively influenced these specific skill sets or that the measurement tools were insufficient. Finally, in the Critical Thinking category, there was a significant improvement in one cohort (Fall 2022). Details of the data analysis, including calculated p-values (p), are provided in Table 1. In the table, 'S' denotes statistically significant change, while 'NS' indicates a non-statistically significant change.

TABLE I
COMPARISON OF SPRING 2022 AND FALL 2022, WITH FALL 2021

Categories	**F-21 vs. SP-22 *(P)***	**F-21 vs. F-22 *(P)***
Creative Thinking	S (0.03)	S (0.03)
Algorithmic Thinking	NS (0.77)	NS (0.22)
Critical Thinking	NS (0.1)	S (0.02)
Problem Solving	S (0.00)	S (0.03)
Confidence	S (0.03)	S (0.02)
Persistent	S (0.04)	S (0.01)
Cooperation	NS (0.18)	NS (0.26)

The findings suggest that while some components of CT, such as critical thinking and problem-solving confidence and persistence in solving problems were significantly enhanced, others, like algorithmic thinking and cooperation, did not show significant improvements.This disparity may necessitate a review and adjustment of the CT modules to specifically target these areas, particularly the Algorithmic Thinking com-

petency. The CT modules could be adjusted to incorporate more collaborative tasks and exercises that build cooperation among students. Additionally, strengthening algorithmic thinking through more targeted activities could balance the development of all CT skills. Furthermore, implementing regular feedback mechanisms within the courses could help identifying areas where students feel less confident or undeserved by the module, allowing for iterative improvements to the curriculum.

IV. Conclusion and Future Work

In this study, we present the results of our intervention involving the design of a set of CT modules as unplugged activities, integrated into a SWE program to enhance students' CT skills. These modules are designed to be generic and do not require specific computing background or programming knowledge to be utilized. This feature enables wider application across multiple engineering and STEM fields. We employed a standard tool to measure the impact of the intervention and the differences in students' CT skills after practicing these modules. The application of these CT modules at a SWE program in an R2 university has demonstrated a significant impact on the development of students' critical thinking and problem-solving skills, confirming the effectiveness of integrating CT into the curriculum. However, the evaluation also identified areas such as algorithmic thinking where improvements were less pronounced. This disparity underscores the necessity for a targeted adjustment to address this skill. Overall, the implementation of the CT modules has been fruitful but highlights the need for ongoing adjustments to maximize their potential in nurturing well-rounded computational thinkers prepared for both academic and professional success.

In the future, we plan to examine the longer-term impact of these interventions on subsequent courses. Furthermore, we intend to disseminate these modules as an independent package to be integrated into Learning Management Systems (LMS) and used by other public universities in engineering programs. We will conduct a multi-institutional analysis to compare the impact of the intervention on different programs. Based on the collected data, we will refine and adjust the modules to better address students' CT skill gaps. Adaptive learning strategies could also be integrated into the CT modules, where real-time adjustments will be made based on student performance and feedback within each module. This could ensure that all students, regardless of their initial skill level, benefit optimally from the CT practices.

Acknowledgment

This intervention was applied in the College of Computing and Software Engineering at Kennesaw State University. We would like to express our gratitude to the FYE Program Director and the faculty members who taught the Introduction to Computing, Programming and Problem Solving I and II courses. Without their support and collaboration in implementing the CT modules and data collection, this research would have not been possible. We also extend our thanks to the Kennesaw State University and the Department of Software Engineering for providing the funding necessary to conduct this intervention. Their commitment to enhancing student success made this research possible and underscores the institution's dedication to educational excellence.

References

[1] Y. A. Rodríguez del Rey, I. N. Cawanga Cambinda, C. Deco, C. Bender, R. Avello-Martínez, and K. O. Villalba-Condori, "Developing computational thinking with a module of solved problems," *Computer Applications in Engineering Education*, vol. 29, no. 3, pp. 506–516, 2021.

[2] J. A. Lyon and A. J. Magana, "Computational thinking in higher education: A review of the literature," *Computer Applications in Engineering Education*, vol. 28, no. 5, pp. 1174–1189, 2020.

[3] J. M. Wing, "Computational thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006.

[4] Y. Li, A. H. Schoenfeld, A. A. diSessa, A. C. Graesser, L. C. Benson, L. D. English, and R. A. Duschl, "On computational thinking and STEM education," *Journal for STEM Education Research*, vol. 3, pp. 147–166, 2020.

[5] N. Dehbozorgi and M. Roopaei, "Improving computational thinking competencies in STEM higher education," in *Proceedings of the 2024 IEEE Integrated STEM Education Conference (ISEC)*, 2024.

[6] PITAC,Computational science: Ensuring America's competitiveness(2005).https://www.nitrd.gov/pitac/reports/20050609

[7] P. Curzon, M. Dorling, T. Ng, C. Selby, and J. Woollard, "Developing computational thinking in the classroom: a framework," Computing at School, 2014.

[8] A. J. Magana and G. Silva Coutinho, "Modeling and simulation practices for a computational thinking-enabled engineering workforce," *Computer Applications in Engineering Education*, vol. 25, no. 1, pp. 62–78, 2017.

[9] J. Malyn-Smith and I. Lee, "Application of the occupational analysis of computational thinking-enabled STEM professionals as a program assessment tool," *Journal of Computational Science Education*, vol. 3, no. 1, pp. 2–10, 2012.

[10] J. F. Sanford and J. T. Naidu, "Mathematical modeling and computational thinking," *Contemporary Issues in Education Research*, vol. 10, no. 2, pp. 159–168, 2017.

[11] D. E. Sondakh, K. Osman, and S. Zainudin, "A proposal for holistic assessment of computational thinking for undergraduate: Content validity," *European Journal of Educational Research*, vol. 9, no. 1, pp. 33–50, 2020.

[12] C. Tikva and E. Tambouris, "A systematic mapping study on teaching and learning computational thinking through programming in higher education," *Thinking Skills and Creativity*, vol. 41, p. 100849, 2021.

[13] C. Cachero, P. Barra, S. Meliá, and O. López, "Impact of programming exposure on the development of computational thinking capabilities: An empirical study," *IEEE Access*, vol. 8, pp. 72316–72325, 2020.

[14] F. J. Agbo, S. S. Oyelere, J. Suhonen, and S. Adewumi, "A systematic review of computational thinking approach for programming education in higher education institutions," in *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*, pp. 1–10, 2019.

[15] I. de Jong and J. Jeuring, "Computational thinking interventions in higher education: A scoping literature review of interventions used to teach computational thinking," in *Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, pp. 1–10, 2020.

[16] J. Bilbao, E. Bravo, O. García, C. Rebollar, and C. Varela, "Study to find out the perception that first year students in engineering have about the computational thinking skills, and to identify possible factors related to the ability of abstraction," *Heliyon*, vol. 7, no. 2, 2021.