Jurnal Teknologi Informasi dan Pendidikan
Volume 18, No. 2, September 2025
https://doi.org/10.24036/jtip.v18i2.977

P.ISSN: 2086 – 4981
E.ISSN: 2620 – 6390
tip.ppj.unp.ac.id

Exploring the Influence of Programming Courses on University Students' Computational Thinking Skills

Rahmadini Darwas[1*]✉, Rina Sepriana[2], Rahimullaily[1], Nizwardi Jalinus[3], Ambiyar[3], Giatman[3]
[1]Universitas Metamedia, Padang, Indonesia
[2]Universitas Putra Indonesia YPTK Padang, Indonesia
[3]Universitas Negeri Padang, Indonesia
**Corresponding Author: dini@metamedia.ac.id*

Article Information

Article history:

No. 977
Rec. June 27, 2025
Rev. July 09, 2025
Acc. July 10, 2025
Pub. July 21, 2025
Page. 302 – 317

Keywords:

- *Computational thinking*
- *Programming course*
- *Correlational study*
- *Learning assesment*

ABSTRACT

This study aims to explore the relationship between students' perceptions of programming courses and computational thinking (CT) skills in Information Systems Study Program students at Metamedia University. The sample consisted of 37 semester students in the 2024/2025 academic year. The research method used was quantitative with a correlational approach, and data analysis was carried out using the JASP application. The results of the Spearman's rho correlation test showed a value of 0.103 with a p-value of 0.590. This value indicates a very weak and insignificant relationship between perceptions of programming courses and CT skills, as measured by the Mid-Semester Exam scores. This finding indicates that although students have positive perceptions of programming courses, these perceptions are not necessarily directly proportional to their CT skills. This result contradicts initial expectations that assume a strong relationship between positive perceptions and increased CT skills. The implications of this study indicate the need to develop a learning approach that explicitly integrates CT elements in the learning process, assignments, and assessments in programming courses in higher education.

How to Cite:

Darwas, R., & et al. (2025). Exploring the Influence of Programming Courses on University Students' Computational Thinking Skills. Jurnal Teknologi Informasi Dan Pendidikan, 18(2), 302-317. https://doi.org/10.24036/jtip.v18i2.977

This open-access article is distributed under the Creative Commons Attribution-ShareAlike 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. ©2023 by Jurnal Teknologi Informasi dan Pendidikan.

1. INTRODUCTION

In the digital era, the integration of computational technologies into nearly every sector of society has significantly reshaped the landscape of education, employment, and daily life [1];[2]. As automation, artificial intelligence, and data-driven decision-making become increasingly prevalent, individuals are expected to possess not only technical knowledge but also the cognitive flexibility to adapt and solve problems in dynamic environments [3]. Consequently, Computational Thinking (CT) has emerged as a critical competence that transcends the boundaries of computer science, becoming a foundational skill necessary for informed participation in the modern world [4]. Educational systems worldwide are now under pressure to embed CT early and meaningfully within curricula to prepare learners for the challenges of the future workforce [5].

The rapid advancement of technology has driven the demand for graduates to possess 21st-century thinking skills, including critical, creative, analytical thinking, and the ability to solve complex problems [6], [7]. Therefore, higher education institutions are expected to produce graduates who are not only proficient in software operation but also possess Computational Thinking (CT) skills [8], [9]. This CT capability is one of the crucial skills that is increasingly under the spotlight along with the demands of the Industrial Revolution 4.0 and Society 5.0 which require individuals with innovative and adaptive problem-solving skills to the dynamics of technological change [10]. This means that today's graduates must be able to break down large problems into smaller parts, identify patterns, develop systematic solutions, and implement them efficiently, not only in the computing domain but also in various disciplines. The integration of CT across disciplines is crucial, as it provides a universal framework for tackling problems regardless of their specific context. CT is not just the ability to program, but rather a thinking process that involves a problem-solving approach in a systematic, logical and structured way, just like how computers process information [11]. CT is a problem-solving approach involving decomposition, abstraction, pattern recognition, and algorithmic thinking [12], [13]. This competency serves as a foundation for addressing complex challenges across fields such as technology, science, and even the humanities. For example in science, CT can help design experiments and analyze big data [14] ; in the humanities [15], CT can be used to analyze texts and understand complex information structures [16]. Beyond these specific applications, CT fosters a mindset of systematic inquiry and iterative refinement, essential for innovation in any field.

Programming courses play a crucial role in the curriculum of higher education, particularly in information technology programs, as they aim to equip students with an understanding of logic, algorithms, and data structures. Effective programming instruction can enhance comprehension of logical structures and build students' confidence in systematically solving technological problems [17], [18], [19]. Through programming, students are trained to think structurally, develop logical steps, and test their solutions, a process that inherently drives the development of CT [20].

CT encompasses a range of cognitive skills, such as decomposition, abstraction, pattern recognition, and algorithm design [21]. It is recognized as a key competency in technology education due to its capacity to help individuals understand and resolve complex problems in various domains. Programming courses are considered a primary instrument for developing students' computational thinking abilities [22]. Previous studies have shown that programming education can foster logical, systematic, and analytical skills [23], [24], thereby laying a strong foundation for CT. Students' perceptions of a course reflect the extent to which they feel able to follow, understand, and benefit from the material being taught [25]. This perception is often used as an early indicator of teaching effectiveness, the quality of the methods applied by lecturers, and the relevance of the material to students' needs and expectations. Several previous studies have shown that students' perceptions of the programming learning experience are related to motivation, satisfaction, and academic outcomes. However, a significant research gap exists; only a few studies have specifically examined the relationship between student`s perceived abilities in programming courses and their computational thinking (CT) skills. The existing studies tend to be narrative in nature and have not quantitatively revealed how programming courses affect students' CT levels. This lack of quantitative evidence creates an empirical void regarding the direct impact of perceived programming proficiency on measurable CT outcomes. On the other hand, there is a perceived gap between the learning process in programming courses and the CT ability results achieved by students. Metamedia University, a private university in Padang City, offers programming courses to its 2nd-semester Information Systems study program students. Despite the curriculum's intent, current problems observed include students' failure to understand basic programming concepts, low active participation in the learning process, and lack of student involvement. These common issues, such as difficulties grasping fundamental programming concepts, limited active participation, and disengagement. Does the programming course significantly impact the development of students' computational thinking abilities? This question becomes particularly salient when considering that a positive learning experience, as reflected in student perceptions, might not always translate into tangible skill acquisition, especially in complex cognitive domains like CT.

Several contemporary studies highlight the importance of integrating explicit pedagogical approaches to foster CT through programming education [17]. Researchers argue that programming instruction does not automatically develop CT skills; instead, it must be systematically designed with an emphasis on reflective practice, open-ended problem-solving, and creative exploration. On the other hand [26], emphasizes that the relationship between programming and CT is complex, as not all programming learning experiences lead to meaningful improvements in computational thinking. An emphasis on careful instructional design and learning strategies that encourage deep thinking, not just syntax, becomes critical to bridging this gap.

P.ISSN: 2086 – 4981
E.ISSN: 2620 – 6390
tip.ppj.unp.ac.id

This phenomenon underscores the critical need for more in-depth quantitative evaluations of how programming education affects key dimensions of CT, such as problem decomposition, pattern recognition, and algorithm formulation. This need aligns with the recommendations of [27] who stress the importance of data-driven assessment in evaluating the effectiveness of programming education, especially in the context of higher education. Furthermore, the current learning environment in Indonesian higher education institutions reveals variability in students' CT outcomes, which may stem from differences in teaching methods, instructor readiness, and students' cognitive backgrounds.

Considering the urgency to strengthen CT in education and the existing gap between instructional practices and student competency outcomes, this study aims to quantitatively explore the influence of programming courses on students' computational thinking abilities. Specifically, this research seeks to address the empirical gap by quantitatively assessing the correlation between students' perceptions of their programming course and their measured computational thinking skills, thereby offering insights into the effectiveness of current educational approaches and informing future pedagogical refinements.

This study was conducted by distributing a questionnaire to 37 students currently enrolled in a programming course. The instrument was developed based on four main indicators: (1) understanding of basic programming concepts and computational thinking; (2) active participation and practical engagement in the learning process; (3) ability to apply CT beyond programming contexts; and (4) students' perceptions of the overall impact of programming courses on their CT development.

2. RESEARCH METHOD

This research was carried out through several main structured stages (shown in Figure 1), including:

2.1. Preparation Stages

At this stage, the main focus is to mature the research foundation, namely:

a. Studying Literature: conducting a comprehensive literature search related to the concept of computational thinking (CT), the influence of programming courses, and relevant measurement instruments. This literature study aims to strengthen the theoretical basis and conceptual framework of the study. Various scientific sources, such as reputable international journals, academic books, and previous research reports are used as references in this process. Thus, a deep understanding of the relationship between perceptions of programming and CT abilities can be developed systematically and based on evidence. This stage ensured that the research design was grounded in existing knowledge and identified potential gaps that this study aimed to address.

b. Designing research instruments: a questionnaire was drafted to measure the perception of programming courses. The drafting of this questionnaire will go through a careful process. Each question in the questionnaire will be strictly adjusted to the research objectives that have been set, ensuring that the data collected is relevant and can answer the research questions. The questionnaire was designed to capture students' perceptions across four main indicators: (1) understanding of basic programming concepts and computational thinking, (2) active participation and practical engagement in the learning process, (3) ability to apply CT beyond programming contexts, and (4) students' perceptions of the overall impact of programming courses on their CT development. The questionnaire utilized a Likert scale with five response options: "Strongly Agree" (5 points), "Agree" (4 points), "Neutral" (3 points), "Disagree" (2 points), and "Strongly Disagree" (1 point).
c. Instrument Validation and Reliability: testing the validity (suitability of the instrument to what will be measured) and reliability (consistency of measurement results) of the designed instrument. Validity test using the Pearson Product Moment correlation statistical method. Reliability test using the Cronbach's Alpha statistical method

2.2. Implementation Stage

This stage is the core of the data collection process, including:

a. Research Socialization: explaining the purpose and procedures of the research to potential respondents (students) to obtain their consent and active participation. This also includes an explanation regarding data confidentiality, ensuring ethical considerations were met.
b. Data collection: questionnaires were distributed online to 37 students currently enrolled in the programming course. This questionnaire was designed to collect information about students' perceptions and experiences regarding the impact of programming courses on their computational thinking skills. Online distribution was chosen for its efficiency, wider reach, and ease for participants to fill out the questionnaire anytime and anywhere, especially considering the constraints of the academic year 2024/2025. The data on computational thinking ability was measured through learning outcomes via the mid-term exam scores for the programming course in the even semester of 2024/2025.

2.3. Data Analysis Stage

Once the data is collected, this stage focuses on data processing and data interpretation, including:

P.ISSN: 2086 – 4981
E.ISSN: 2620 – 6390
tip.ppj.unp.ac.id

a. Data input: entering all collected data according to research needs into JASP statistical software. This step ensured that the dataset was appropriately structured for subsequent statistical analyses.
b. Normality Test as a requirement for conducting parametric correlation analysis (Pearson correlation). If the variables are not normally distributed then to analyze the correlation using non-parametric correlation (Spearman's Rho and Kendall's tau-b).
c. Correlation Test, Correlation test is a statistical technique used to determine the relationship or connection between two variables. This test does not show a cause-and-effect relationship, but only measures the extent to which changes in one variable are related to changes in another variable. The correlation test in this study was used to measure the strength and direction of the relationship between perceptions of programming courses and computational thinking skills.

Given this study only includes one independent variable (perception of programming courses), multicollinearity diagnostics were not strictly required. However, to ensure no redundancy among composite perception indicators, VIF was computed as an additional check.

2.4. Final Stage

This final stage includes:

a. Interpretation of results: analyzing and interpreting findings from data analysis relating them to existing theories and the literature reviewed.
b. Discussion: discusses the implications of the research results, explains and relates them to the problem formulation and research objectives.
c. Conclusions and suggestions: formulate conclusions based on research findings and provide relevant suggestions for curriculum development, teaching methods, or further research.

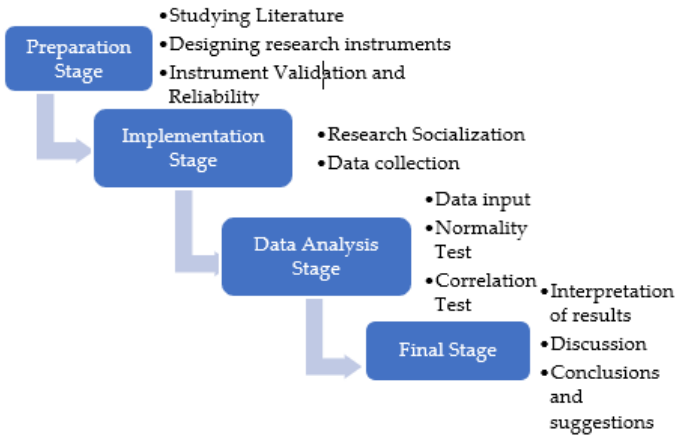


Figure 1. Research Stages

This study is quantitative research using a correlational approach to examine the effect of programming courses on students' computational thinking skills. The population of this research consists of students from the Information Systems Study Program at Universitas Metamedia who were enrolled in a programming course during the even semester of the 2024/2025 academic year. A purposive sample of 37 students currently taking the course was selected for this study. Sampling was conducted using purposive sampling technique, where the researcher selected subjects based on certain criteria that were considered relevant to the research objectives, so that 37 students were selected from the current semester. The selection of this purposive sampling method was based on the consideration that students who were taking programming courses directly were the most appropriate parties to provide perceptions about the course and their CT abilities could be measured in a relevant learning context, so that the data obtained were more focused and in accordance with the research problems. Due to the limited number of students enrolled in the programming course during the even semester of 2024/2025, the sample size was restricted to 37 participants. This study involves two variables:

1. Independent Variable (X): Programming Course, which refers to students' erceptions in taking programming courses, which is measured through a Likert-scale questionnaire. The Likert scale is used to measure the level of agreement.
2. Dependent Variable (Y): Computational thinking ability, which refers to students' capacity to solve problems using computational thinking concepts, which is measured through learning outcomes through the mid-term exam scores for the programming course in the even semester of 2024/2025.

The instrument used in this research was a questionnaire consisting of four main indicators:

1. Understanding of Basic Programming and CT Concepts
    a. I find it easy to understand basic programming concepts (e.g., variables, control structures, functions) taught in this course.
    b. This programming course helps me break down complex problems into smaller components (decomposition).
    c. I can recognize patterns or similarities in various programming problems after taking this course (pattern recognition).
    d. This course has equipped me with the ability to formulate systematic steps or algorithms to solve problems.
    e. I am able to create models or abstractions of systems or problems in order to solve them (abstraction).
    f. I understand how to debug (identify and fix errors in) my code.
2. Active Participation and Practical Engagement
    a. I feel motivated to actively participate in discussions and activities in this programming course.

P.ISSN: 2086 – 4981
E.ISSN: 2620 – 6390
tip.ppj.unp.ac.id

b. The practical assignments in this course are challenging and encourage me to experiment with code.
c. I feel that I have sufficient opportunities to apply programming theory in practical projects.
d. The lecturer/teaching assistant provides constructive feedback on my code or programming solutions.
e. I often discuss or collaborate with peers to solve programming problems.

3. Application of Computational Thinking Beyond Programming Context
   a. I feel my problem-solving abilities outside programming contexts have improved after taking this course.
   b. I can apply logic and systematic thinking gained from programming to solve everyday problems.
   c. This programming course helps me think in a more structured and logical way in various situations.
   d. I feel more confident in approaching new problems using a systematic thinking pattern.
4. Overall Impact of the Programming Course
   a. This programming course has significantly improved my computational thinking skills.
   b. I believe that the curriculum and teaching methods in this programming course are effective in developing CT.
   c. I would recommend this programming course as an essential course for developing computational thinking skills.

The questionnaire was presented using a Likert scale, which is a commonly used measurement scale for assessing attitudes or opinions. It consisted of five response options, as shown in Table 1.

Table 1. The Likert scale ranges

Statement	Point
Strongly Agree	5
Agree	4
Neutral	3
Disagree	2
Strongly Disagree	1

Test of instrument validity and reliability testing were conducted using the JASP application. Validity test, which aims to measure the extent to which the questionnaire is able to measure what should be measured, is done by comparing the r-count value (Pearson's correlation coefficient between item scores and total scores) with the r-table value. With degrees of freedom (df) of n-2 (37 - 2 = 5) and a significance level (sig) of 0.05, the r-table is 0.334. In accordance with the established criteria, a statement item is considered

valid if its r-count value is greater than the r-table value [28], indicating that the item contributes significantly to the measurement of the construct being studied and is suitable for use in data collection. The results of the validity test are presented in Table 2 where all statement items show a Pearson correlation coefficient value (R Count) that is greater than R Table (0.334), so that all items are declared valid.

Table 2. Validity Test

No Item	Pearson Correlation Coefficient Value		Information
	R Count	R Table	
1	0.724	0.334	Valid
2	0,624	0.334	Valid
3	0.618	0.334	Valid
4	0.818	0.334	Valid
5	0.684	0.334	Valid
6	0.774	0.334	Valid
7	0.577	0.334	Valid
8	0.722	0.334	Valid
9	0.771	0.334	Valid
10	0.798	0.334	Valid
11	0.835	0.334	Valid
12	0.665	0.334	Valid
13	0.681	0.334	Valid
14	0.625	0.334	Valid
15	0.701	0.334	Valid
16	0.663	0.334	Valid
17	0.685	0.334	Valid
18	0.848	0.334	Valid

Source: Researcher's processing

Next, the reliability test is conducted to assess the internal consistency of the questionnaire instrument. Reliability measurement uses Cronbach's Alpha coefficient. An instrument is considered reliable if it has a high Cronbach's Alpha value, which indicates that the items in the questionnaire consistently measure the same construct. The reliability test was measured using Cronbach's Alpha. The Cronbach's Alpha value obtained was 0.972, indicating that the instrument has a very high level of internal consistency [29]. The results of the reliability test are shown in Table 3.

Table 3. Frequentist scale reliability statistics

Estimate	Cronbach`s α
Point estimate	0.972

P.ISSN: 2086 – 4981
E.ISSN: 2620 – 6390
tip.ppj.unp.ac.id

The data obtained from all research samples, including perceptions of programming ability and computational thinking outcomes, were statistically analyzed. Perceptions of programming ability were measured using a Likert-scale questionnaire, while computational thinking performance was assessed based on the midterm exam scores of students enrolled in the Programming Course during the Even Semester of the 2024/2025 academic year. The Variance Inflation Factor (VIF) analysis showed that all indicators of students' perceptions had VIF values below 5, suggesting that multicollinearity was not a concern in this study.

The initial step of the analysis involved conducting a Shapiro-Wilk normality test for each variable using the JASP software. If the variables in this study are normally distributed, the Pearson Correlation Test statistical method is used to measure the strength and direction of the relationship between the independent and dependent variables. However, if the research variables are not normally distributed, the Spearman's Rho non-parametric correlation test statistical method and Kendall's tau-b non-parametric correlation are used. All statistical analyses in this study were performed at a significance level of $\alpha = 0.05$. The purpose of this analysis was to measure the strength and direction of the relationship between students' perceived abilities in programming and their computational thinking skills.

3. RESULTS AND DISCUSSION

This section explains the research results and, at the same time, a comprehensive discussion. Results can be presented in figures, graphs, tables, and others that make the reader understand easily [14], [15]. The discussion can be made in several sub-sections.

The normality test was conducted to determine whether the data are normally distributed, ensuring that the statistical methods used are appropriate and yield accurate conclusions [30]. Variable X represents the perception of programming ability, while Variable Y represents the outcome of students' computational thinking skills. The results of the Shapiro-Wilk normality test are presented in Table 4.

Table 4. Normality test

	Perception of Programming Ability	Computational Thinking Ability Results
Valid	37	37
Missing	0	0
Mean	66.784	64.811
Std. Deviation	12.802	22.378
Kurtosis	4.908	1.993
Std. Error of Kurtosis	0.759	0.759
Shapiro-Wilk	0.903	0.782
P-value of Shapiro-Wilk	0.004	< .001
Minimum	18.000	0.000

Jurnal Teknologi Informasi dan Pendidikan
Volume 18, No. 2, September 2025
https://doi.org/10.24036/jtip.v18i2.977

Table 4. Normality test

	Perception of Programming Ability	Computational Thinking Ability Results
Maximum	90.000	90.000

Based on Table 2, the p-values for the variables of perceived programming ability and computational thinking skills were 0.004 and $p < 0.001$, respectively. Both p-values indicate that the two variables are not normally distributed (since both p-values < 0.05). This is further supported by the negative skewness values (-1.424 and -1.554) and relatively high kurtosis values (4.908 and 1.993). Since the assumption of normality is not met and the data are on a ratio scale, the relationship between the two variables was analyzed using non-parametric correlation tests, specifically Spearman's Rho and Kendall's tau-b correlation Spearman`s rho correlation test is shown in Table 5.

Table 5. Spearman`s rho correlation test

Variable		Perception of Programming Ability	Computational Thinking Ability Results
1. Perception of Programming Ability	Spearman's rho	—	
	p-value	—	
2. Computational Thinking Ability Results	Spearman's rho	0.103	—
	p-value	0.545	—

Based on Table 5, the Spearman's rho correlation coefficient is 0.103, indicating a very weak and positive relationship between the programming course and computational thinking skills. This means that as students' perceptions of the programming course improve, their computational thinking skills tend to increase slightly, although the correlation is very weak.

This finding is also supported by the p-value, which is 0.545 (p-value = 0.545). Since this p-value is greater than 0.05, there is insufficient evidence to conclude that there is a significant relationship between students' perceived competence in the programming course and their computational thinking ability. Kendall's tau-b correlation test is shown in Table 6.

Table 6. Kendall's tau-b correlation test

Variable		Perception of Programming Ability	Computational Thinking Ability Results
1. Perception of Programming Ability	Kendall's Tau B	—	

P.ISSN: 2086 – 4981
E.ISSN: 2620 – 6390
tip.ppj.unp.ac.id

Table 6. Kendall's tau-b correlation test

Variable		Perception of Programming Ability	Computational Thinking Ability Results
	p-value	—	
2. Computational Thinking Ability Results	Kendall's Tau B	0.066	—
	p-value	0.590	—

Based on Table 6, the Kendall's Tau-b correlation coefficient was found to be 0.066, indicating a very weak and positive relationship between the two variables. This means that as students' perceptions of the programming course increase, their computational thinking skills tend to slightly improve; however, the correlation is very weak. This is also supported by the p-value of 0.590 (p-value = 0.590). Since the p-value is greater than 0.05, the relationship is not statistically significant. In other words, there is insufficient evidence to conclude that a meaningful relationship exists between students' perceptions of the programming course and their computational thinking skills.

This study shows that although students have positive perceptions of programming courses, these perceptions do not directly correlate significantly with computational thinking (CT) skills as measured by Mid-Semester Exam (UTS) scores. This finding is somewhat contrary to initial expectations that assumed that positive perceptions of the programming learning experience would directly translate into measurable improvements in CT abilities. Several possibilities that can explain this, which need to be explored further to understand the dynamics of the relationship between programming learning and CT development, include:

a. Mid-term exam scores may not yet holistically reflect students' computational thinking abilities. It is possible that exam questions, especially at the basic level, tend to emphasize memorization or programming syntax, remembering basic rules, rather than CT logic such as problem decomposition (breaking down complex problems into smaller parts), abstraction (identifying the essence of a problem without being burdened with details), and algorithm design (arranging logical steps to solve a problem). Although mid-term exam scores were used as a proxy for CT skills, they may emphasize code correctness over CT aspects like abstraction or decomposition. Future research should incorporate direct CT assessments or performance tasks. If the assessment focuses more on the correct code output rather than the thinking process behind it, then the mid-term exam scores will not be an accurate proxy for CT ability.
b. High perceptions do not always align with actual competence or academic performance, especially if the learning process does not fully support the real practice of computational thinking. Students may feel that programming courses are important and beneficial, but this feeling does not necessarily mean that they have fully internalized and are able to apply CT concepts in real practice. This gap between

perception and competence can occur if the learning process does not fully support the real practice of computational thinking explicitly. For example, if lectures are more dominant in lectures or exercises that do not encourage exploration of complex and creative problems, positive perceptions can still exist without being balanced by significant improvements in CT abilities.

c. There are other external factors that may play a role that are not explicitly measured in this questionnaire. Factors such as learning motivation, previous programming experience (e.g., online courses, projects, or other activities), or non-formal learning support (e.g., developer communities, mentors, or self-study resources outside of formal lectures) may significantly influence the development of CT skills. Students with high motivation and additional experience outside of class may demonstrate better CT skills regardless of their perceptions of formal courses. Therefore, the relationship between formal programming courses and CT may be moderated by these external variables.

These results provide important implications for the development of programming curricula. A more explicit learning approach is needed to integrate CT dimensions into lectures, assignments, and assessments. Evaluation also needs to be developed to assess not only technical aspects, but also logic and problem-solving strategies. This means moving beyond simple code execution to assess how students decompose problems, identify patterns, and design efficient algorithms. This study has several limitations. First, the small sample size (n=37) limits the generalizability of the findings. Second, computational thinking was measured using mid-term exam scores, which may not fully capture all CT dimensions such as abstraction, decomposition, or algorithm design. Third, external factors (e.g., motivation, prior experience) were not controlled for and could influence CT outcomes. For future research, it is recommended to:

a. Develop More Robust CT Assessment Tools: Future studies should utilize or develop more comprehensive CT assessment tools that move beyond traditional exam scores, such as performance-based tasks, coding challenges requiring complex problem-solving, or rubric-based evaluations of students' algorithmic thinking and decomposition skills. This would provide a more accurate measure of actual CT ability.
b. Investigate Pedagogical Interventions: Explore the effectiveness of specific pedagogical interventions aimed at explicitly fostering CT within programming courses. This could involve adopting problem-based learning, project-based learning, or inquiry-based learning approaches that emphasize the CT components.
c. Incorporate Qualitative Data: Supplement quantitative findings with qualitative data, such as interviews or focus group discussions with students and instructors, to gain deeper insights into their perceptions, challenges, and successful strategies in developing CT skills through programming.
d. Control for Confounding Variables: Future studies should aim to measure and control for external factors like prior programming experience, self-regulated learning

P.ISSN: 2086 – 4981
E.ISSN: 2620 – 6390
tip.ppj.unp.ac.id

strategies, and motivation, which might mediate the relationship between perceived course effectiveness and CT outcomes. Future studies are encouraged to include mediation or moderation analysis to investigate the influence of external factors such as motivation, prior experience, and informal learning.

e. Longitudinal Studies: Conduct longitudinal studies to observe the development of CT skills over a longer period, perhaps across multiple programming courses, to capture the cumulative impact and maturation of these skills.

By addressing these points, future research can provide a more nuanced and complete understanding of how programming education contributes to the crucial development of computational thinking skills in university students. Theoretically, this study adds to the limited body of quantitative evidence on the relationship between programming courses and computational thinking skills. Practically, the findings suggest that course design should explicitly embed CT components beyond syntax and code correctness, especially for industry contexts where problem decomposition and algorithm design are highly valued.

4. CONCLUSION

This study aims to explore the influence of the programming course on students' computational thinking (CT) abilities using a correlational approach. The validity and reliability tests of the questionnaire indicated that the instrument used had excellent measurement quality (p-value < 0.01; Cronbach's alpha = 0.972). However, the results of Spearman's Rho and Kendall's tau-b correlation analyses showed that the relationship between student`s perceived programming skills in the programming course and their computational thinking ability (measured by the midterm exam scores as a proxy for CT skills) was not statistically significant (p-values for both methods > 0.05).

These findings indicate that a positive perception of the programming learning process does not necessarily impact computational thinking abilities as measured academically. This implies the need for a reevaluation of the teaching approach and assessment methods in the programming course to better align with the development of essential computational thinking skills.

REFERENCES

[1] M. Hilbert, "Digital technology and social change: the digital transformation of society from a historical perspective," *Dialogues Clin. Neurosci.*, vol. 22, no. 2, Jun. 2020, doi: 10.31887/DCNS.2020.22.2/mhilbert.

[2] A. Shehaj, "Revolutionized learning: Education policy and digital reform in the eurozone," *Eur. Policy Anal.*, vol. 8, no. 3, Jun. 2022, doi: https://doi.org/10.1002/epa2.1158.

[3] A. Siqi-Liu and T. Egner, "Contextual Adaptation of Cognitive Flexibility is driven by Task- and Item-Level Learning," *Cogn. Affect. Behav. Neurosci.*, vol. 20, no. 4, pp. 757–782, 2020, doi: 10.3758/s13415-020-00801-9.

[4] B. Rienties, "Defining the Boundaries Between Artificial Intelligence in Education, Computer-Supported Collaborative Learning, Educational Data Mining, and Learning Analytics: A Need for Coherence," 2020. doi: 10.3389/feduc.2020.00128.
[5] I. W. Widiana, I. W. Lasmawan, and I. G. P. Suharta, "Curriculum Transformation Towards Future Education," *Prism. Magistra J. Ilm. Kependidikan*, vol. 6, no. 2, pp. 122–132, 2025.
[6] R. Darwas, R. Sepriana, A. Yulastri, D. Irfan, and N. Jalinus, "Pengaruh Project Based Learning Terhadap Kemampuan Berpikir Peserta Didik: Studi Meta Analisis," *Nusant. J. Pendidik. Indones.*, vol. 5, no. 1, 2025, doi: https://doi.org/10.62491/njpi.2025.v5i1-3.
[7] R. H. Mardhiyah, S. N. F. Aldriani, F. Chitta, and M. R. Zulfikar, "Pentingnya Keterampilan Belajar di Abad 21 sebagai Tuntutan dalam Pengembangan Sumber Daya Manusia," *Lect. J. Pendidik.*, vol. 12, no. 1, pp. 29–40, May 2021, doi: 10.31849/lectura.v12i1.5813.
[8] A. A. Ogegbo and R. Umesh, "A Systematic Review of Computational Thinking in Science Classrooms," 2022. doi: 10.1080/03057267.2021.1963580.
[9] 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," *Comput. Appl. Eng. Educ.*, vol. 29, no. 3, pp. 506–516, May 2021, doi: 10.1002/CAE.22214')).
[10] S. Bartoloni *et al.*, "Towards designing society 5.0 solutions: The new Quintuple Helix - Design Thinking approach to technology," *Technovation*, vol. 113, p. 102413, May 2022, doi: 10.1016/J.TECHNOVATION.2021.102413.
[11] N. O. Ezeamuzie and J. S. C. Leung, "Computational Thinking Through an Empirical Lens: A Systematic Review of Literature," *J. Educ. Comput. Res.*, vol. 60, no. 2, pp. 481–511, 2021, doi: https://doi.org/10.1177/07356331211033158.
[12] A. Sjödahl and A. Eckert, "Abstracting and decomposing in a visual programming environment - ScienceDirect," *Int. J. Child-Computer Interact.*, vol. 36, May 2023, doi: https://doi.org/10.1016/j.ijcci.2023.100573.
[13] D. Weintrop, M. Shandra, and M. and Subramaniam, "Assessing computational thinking in libraries," *Comput. Sci. Educ.*, vol. 31, no. 2, pp. 290–311, May 2021, doi: 10.1080/08993408.2021.1874229.
[14] I. Lee, "Computational Thinking from a Disciplinary Perspective: Integrating Computational Thinking in K-12 Science, Technology, Engineering, and Mathematics Education," *J. Sci. Educ. Technol.*, vol. 29, no. 1, pp. 1–8, 2020, doi: 10.1007/s10956-019-09803-w.
[15] Z. Katai, "Promoting computational thinking of both sciences- and humanities-oriented students: an instructional and motivational design perspective," *Educ. Technol. Res. Dev.*, vol. 68, no. 5, pp. 2239–2261, 2020, doi: 10.1007/s11423-020-09766-5.
[16] Y. Li, "Computational Thinking Is More about Thinking than Computing," 2020. doi: 10.1007/s41979-020-00030-2.
[17] W. Deng, "Pencil Code improves learners' computational thinking and computer learning attitude," *Comput. Appl. Eng. Educ.*, vol. 28, no. 1, pp. 90–104, 2020, doi: 10.1002/cae.22177.
[18] Y.-F. Liu, J. Kim, C. Wilson, and M. Bedny, "Computer code comprehension shares neural resources with formal logical inference in the fronto-parietal network," *Elife*, vol. 9, p. e59340, May 2020, doi: 10.7554/eLife.59340.
[19] Y. Zhuang, L. Yu-Hsuan, L. Mahesh, S. Andito Haryo, U. Yuniati Dwi, and J.-H. and Wang, "An interactive programming learning environment supporting paper computing and immediate

evaluation for making thinking visible and traceable," *Interact. Learn. Environ.*, vol. 32, no. 9, pp. 5253–5266, May 2024, doi: 10.1080/10494820.2023.2212709.

[20] G. Mecca, D. Santoro, N. Sileno, and E. Veltri, "Diogene-CT: tools and methodologies for teaching and learning coding," *Int. J. Educ. Technol. High. Educ.*, vol. 18, no. 1, pp. 1–26, 2021, doi: 10.1186/s41239-021-00246-1.

[21] N. Pellas, "Enhancing Computational Thinking, Spatial Reasoning, and Executive Function Skills: The Impact of Tangible Programming Tools in Early Childhood and Across Different Learner Stages," *J. Educ. Comput. Res.*, vol. 63, no. 1, 2025, doi: https://doi.org/10.1177/07356331241292767.

[22] 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, May 2020, doi: 10.1109/ACCESS.2020.2987254.

[23] R. C. Coelho, M. F. P. Marques, and T. de Oliveira, "Mobile Learning Tools to Support in Teaching Programming Logic and Design: A Systematic Literature Review," *Informatics Educ.*, vol. 22, no. 4, pp. 589–612, May 2023, doi: 10.15388/infedu.2023.24.

[24] C. Y. Tsai and Y. C. Lai, "Design and Validation of an Augmented Reality Teaching System for Primary Logic Programming Education," *Sensors*, vol. 22, no. 1, May 2022, doi: https://doi.org/10.3390/s22010389.

[25] C. M. Amerstorfer and C. Freiin von Münster-Kistner, "Student Perceptions of Academic Engagement and Student-Teacher Relationships in Problem-Based Learning," *Front. Psychol.*, vol. 12, no. October, pp. 1–18, 2021, doi: 10.3389/fpsyg.2021.713057.

[26] N. O. Ezeamuzie, "Abstractive-Based Programming Approach to Computational Thinking: Discover, Extract, Create, and Assemble," *J. Educ. Comput. Res.*, vol. 61, no. 3, May 2023, doi: https://doi.org/10.1177/07356331221134.

[27] W. Xing and X. Wang, "Understanding students' effective use of data in the age of big data in higher education: Behaviour & Information Technology: Vol 41 , No 12 - Get Access," May 29, 2022. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/0144929X.2021.1936176

[28] Z. Guo, "Causal inference with invalid instruments: post-selection problems and a solution using searching and sampling," *J. R. Stat. Soc.*, vol. 85, no. 3, May 2023, doi: https://doi.org/10.1093/jrsssb/qkad049.

[29] I. W. Gunada, W. Jufri, and S. Idawati, "Validitas dan Reliabilitas Instrumen Pemahaman Nature Of Science Mahasiswa Calon Guru," *J. Pendidikan, Sains, Geol. dan Geofis.*, vol. 6, no. 2, 2025, doi: https://doi.org/10.29303/Goescienceed.v6i2.1041.

[30] R. Purwasih, R. Rahimullaily, and A. I. Suryani, "Blended Learning Model in Improving 4C Abilities of Information System Students," *JPI (Jurnal Pendidik. Indones.*, vol. 10, no. 4, May 2021, doi: 10.23887/jpi-undiksha.v10i4.30939.