Voice of Academia

Academic Series of Universiti Teknologi MARA Kedah

VOLUME 22
ISSUE 1
2026
eISSN: : 2682-7840

e-ISSN:2682-7840

© Voice of Academia is jointly published by the Universiti Teknologi MARA Caawangan Kedah, Malaysia and Penerbit UiTM (UiTM Press), Universiti Teknologi MARA Malaysia, Shah Alam, Selangor.

TABLE of CONTENTS

Voice of Academia
Academic Series of Universiti Teknologi MARA Kedah

Voice of Academia Vol.22 Issue (1) 2026

Voice of Academia

e-ISSN: 2682-7840

COMPARATIVE ANALYSIS OF PSEUDOCODE AND FLOWCHARTS IN ALGORITHM DEVELOPMENT AMONG FIRST-YEAR COMPUTER SCIENCE STUDENTS

Satria Arjuna bin Julaihi[1], Zubaidah binti Bohari[2]*, Rumaizah binti Che Md Nor[3] & Abdul Hadi bin Abdul Talip[4]

[1,2,3,4] *Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA Cawangan Sarawak, Kampus Samarahan 2*

ARTICLE INFO

Article history:

Received Aug 2025
Accepted Oct 2025
Published Jan 2026

Keywords:

<Pseudocode, Flowcharts, Algorithm development, Programming Education, Computational Thinking>

Corresponding Author:
*zubaidah@uitm.edu.my

ABSTRACT

Algorithm development is a fundamental skill in computer science education, yet students struggle to translate problem-solving into structured logic. Flowcharts and pseudocode support algorithmic thinking, but limited research has examined how students perceive and prefer these methods. This study explores first-year computer science students' perceptions and preferences about using flowcharts, pseudocode, or both in algorithm development, and examines how effective these tools are on students' perceptions. 95 students learned both tools via instruction and exercises, then their perceptions were surveyed. Data analysis included descriptive stats, t-tests, ANOVA, and regression. Results showed no significant perception difference ($p > 0.05$), indicating equal value for both tools. Regression analysis further showed that the perceived effectiveness of both tools significantly contributed to the students' perception in learning algorithms development (p-value < 0.05). The study concludes that either approach can effectively support diverse learning styles and enhance algorithmic thinking. It is recommended that future research explore the use of other tools for algorithm development, such as algorithm animation software, block-based programming environments, or interactive visualization platforms, and investigate their long-term impact on programming performance and retention.

1. Introduction

The understanding and application of algorithms are critical components of computer science education, serving as the foundation for the development of programming and computational problem-solving skills. For first-year computer science students, mastering the structuring of algorithms is often a challenging but essential task, as they are required to convert real-life problems into sequences of logical operations that can be executed through code (Liu et al., 2024). Educators frequently employ instructional tools like pseudocode and flowcharts to assist students in addressing this challenge by providing structured yet flexible representations of algorithms (Tarsini & Anggraeni, 2024). Flowcharts, through their use of graphical symbols to depict processes, often particularly appeal to visual learners by presenting information in an easily digestible format (Sagala & Yahfizham, 2024). Concurrently, pseudocode mirrors programming syntax through a structured, natural language approach, acting as a transitional tool that bridges abstract logic with practical coding skills (Xu et al., 2024).

Although both methods are widely used, there is a lack of empirical comparisons regarding their effectiveness for novice learners. Current assessments of these teaching methods highlight a gap in comprehensive studies that examine their effectiveness for beginners. Some research indicates that flowcharts can facilitate quicker comprehension and reduced error rates due to their visual nature, benefiting those who learn more effectively from graphical representations (Andrzejewska & Stolińska, 2022). However, other studies suggest that while flowcharts may offer immediate cognitive advantages, they can hinder learning when students face complex programming problems, where a text-based approach like pseudocode provides a clearer pathway to writing executable code (Yu & Bozic, 2023). Furthermore, existing literature often overlooks the personal preferences and perceptions of students regarding these tools, focusing predominantly on quantitative metrics such as performance measurements rather than qualitative insights that could inform more effective teaching strategies (Gao et al., 2025).

Recognizing the diversity of learning styles among students, the present study seeks to investigate the subjective experiences and preferences of first-year computer science students regarding pseudocode and flowcharts. The findings aim to deepen educators' understanding of how these tools facilitate learning, ultimately guiding curricular decisions that accommodate various learning modalities and foster stronger algorithmic thinking skills.

While several studies have investigated the effectiveness of these tools in programming education, a significant gap remains regarding students' direct perceptions and preferences. Many existing studies emphasize pedagogical impacts rather than capturing the learners' subjective experiences and their perceptions of the tools' effectiveness (Grawemeyer et al., 2022). Gaining insight into how students perceive these tools can provide valuable guidance for educators seeking to adapt instructional methods to accommodate diverse learning needs and support different styles. Therefore, this study aims to address this gap by exploring the perceptions and preferences of first-year computer science students regarding flowcharts and pseudocode in the context of algorithm development. In addition, the study also assesses the perceived effectiveness of these tools in students' learning journeys.

As this discussion has shown, algorithm development remains a fundamental skill in computer science education, requiring effective strategies to support first-year students in their cognitive journey from abstract problem-solving to concrete code implementation. Pseudocode and flowcharts present valuable yet distinct pedagogical tools. This study aims to bridge the existing research gap regarding students as novice programmers and their perceptions and preferences of algorithmic tools, ultimately contributing practical recommendations for educators on how to best enhance these instructional methodologies to foster stronger algorithmic thinking.

The objectives of this study are:

- To investigate the comparative perceptions of first-year students regarding pseudocode and flowcharts as tools for algorithm development.
- To evaluate the effectiveness of these tools on the perceptions of first-year computer science students.

2. Literature Review

2.1 Algorithm Building as The Foundation of Computational Thinking

Computational thinking (CT) has emerged as a critical educational skill in the digital era, extending beyond computer science to include systematic problem-solving methods that are applicable across fields and disciplines. CT involves several core elements: Decomposition, which simplifies complex problems into smaller, manageable parts; Pattern Recognition, which detects similarities and trends in data to aid solutions; Abstractions, which identify patterns and omit unnecessary details; and Algorithmic Thinking, which includes creating a step-by-step set of instructions (an algorithm) to solve problems. These components are essential for developing solutions to complex challenges (Wing, 2006). At its core, CT is about developing algorithms, translating real-world problems into structured sequences of logical steps that computers can follow. This translation often proves difficult for first-year students, who may find the abstraction necessary to convert problems into formal logical structures for programming quite challenging (Kurniawan et al., 2021).

At the core of computational thinking is the art of algorithm development, where individuals learn to convert real-world problems into structured sequences of logical steps that computers can understand and execute. This process not only demands a deep understanding of the problem but also the ability to express it as precise, formal instructions. Unfortunately, this translation can be quite challenging for first-year students, who often struggle to grasp the level of abstraction required to turn their intuitive problem-solving methods into formal logical constructs suitable for programming (Kurniawan et al., 2021). This difficulty underscores the importance of educational frameworks that support the development of these essential skills, helping students bridge the gap between conceptual understanding and practical implementation in computational settings.

2.2 Pseudocode and Flowcharts as Pedagogical Tools in Programming Education

To address this challenge, introductory programming courses widely adopt two pedagogical tools, namely pseudocode and flowcharts. Pseudocode is a written method combining natural language and simplified syntax to describe algorithms. Its main advantage is its flexibility, as it is not constrained by the syntax of a specific programming language, enabling students to concentrate on algorithm logic and control flow (Xu et al., 2024). It acts as a direct, language-independent blueprint for the final code, serving as a helpful intermediary step towards establishing the final coding (Xu et al., 2024).

On the other hand, flowcharts visually represent algorithms, using standardized symbols and connecting lines to illustrate steps, processes, decisions, and loops. Advocates believe that flowcharts' visual style makes them especially intuitive for beginners and visual learners, offering a straightforward map of program logic. This clarity helps identify and fix logical errors early, before coding begins (Andrzejewska & Stolińska, 2022). Zhou et al.(2024) support this by indicating that students better understand algorithms when they are presented visually.

2.3 Empirical Comparisons of Pseudocode and Flowcharts in Programming Education

Several studies have examined the empirical differences between pseudocode and flowcharts, offering insights into their effectiveness as teaching tools in programming education. These studies highlight both the strengths and weaknesses of each method, helping to clarify how they can improve learning outcomes, especially for beginners. Although the studies vary in their findings regarding the impact and usefulness of these approaches, they collectively contribute to a better understanding of their educational roles.

A study conducted by Andrzejewska and Stolińska (2022) demonstrates the effectiveness of flowcharts in enhancing problem-solving skills among novice learners. By employing eye-tracking technology, the research indicates that students can efficiently process and analyze flowcharts with fewer errors in comparison to pseudocode, thereby suggesting a cognitive advantage of visual representation. On the other hand, Richter et al. (2022) argued that the limitations of flowcharts, especially their cumbersome nature when addressing complex problems, are considerable. The authors assert that while flowcharts provide clarity for simple algorithms, they may become inefficient for large-scale or multi-layered algorithms, thereby rendering pseudocode a more suitable format in such cases. In a critical analysis conducted by Al-Fedaghi (2025), he addresses the deceptive simplicity of flowcharts, arguing that their graphical nature may fail to capture the complexities inherent in real-world programming challenges. This critique invites educators to carefully consider the context in which they employ flowcharts versus pseudocode.

Regardless of these arguments, a substantial body of research shows that the choice between the two tools may not be a simple zero-sum game. Grawemeyer et al. (2022) state that the effectiveness of instructional tools, including pseudocode and flowcharts, heavily depends on how they are integrated into the curriculum. It emphasizes the importance of feedback mechanisms and student engagement in determining their success, suggesting that both flowcharts and pseudocode can be equally valuable when used effectively. Additionally, Xu et al. (2024) highlight the flexibility of pseudocode as a language-independent tool that helps students focus on logic and control flow without being limited by the syntax of specific programming languages. By demonstrating its usefulness, the study supports the notion that pseudocode can complement flowcharts in programming education.

Other studies examine the broader educational context in which both tools are used, highlighting the challenges first-year students face when moving from abstract ideas to practical programming applications. This emphasizes the importance of practical teaching tools to facilitate this transition (Kurniawan et al., 2021). Additionally, a study investigates students' preferences and experiences with pseudocode and flowcharts. This research offers qualitative insights that underscore the significance of understanding learners' subjective experiences in programming education (Gao et al., 2025).

By integrating these references, researchers and educators can better understand the implications of using pseudocode and flowcharts in programming education, recognizing their unique strengths and possible limitations in enhancing student learning outcomes. Further empirical research could continue to improve our understanding of how to effectively incorporate these tools into educational settings.

2.4 Cognitive Load and Learning Theories

The effectiveness of pseudocode and flowcharts in designing algorithms can be understood through Cognitive Load Theory (CLT). CLT states that working memory has limited capacity, so instructional tools should minimize unnecessary processing to help learners focus on key ideas (Sweller, 1988; Mayer, 2004). Flowcharts, with their visual overview of processes, can help lessen extraneous cognitive load, allowing beginners to quickly grasp logical connections. In contrast, pseudocode may initially impose a higher intrinsic load because it involves reading and interpreting structured text. However, pseudocode more effectively supports germane load, assisting students in forming schemas that are applicable to real programming. This indicates that the two tools may offer different types of cognitive support.

Another viewpoint is offered by Dual Coding Theory (Paivio, 1990), which highlights that information is processed via two interconnected systems: the verbal and the visual. Flowcharts utilize the visual system, while pseudocode activates the verbal or textual system. When used together, they create a redundant coding of algorithmic ideas, enhancing understanding and memory. Studies have shown that instructional designs that combine both visual and textual modes lead to better retention and fewer misunderstandings than text-only methods (Clark & Paivio, 1991; Mayer, 2009). This suggests that pseudocode and flowcharts are not competing tools but complementary strategies that align with human cognitive architecture.

From a constructivist learning perspective, pseudocode and flowcharts serve as scaffolding tools that help learners develop mental models of algorithms. Constructivism argues that learners actively create knowledge by linking new information to their existing understanding (Bruner, 1961; Vygotsky, 1978). Flowcharts enable students to visually represent their reasoning, making it easier to spot logical errors and consider the structure of an algorithm. Pseudocode, by contrast, involves expressing control flow and conditions in a structured, natural language, which promotes deeper involvement in algorithm design. Research in programming education indicates that these scaffolding methods help bridge the gap between abstract ideas and practical coding, aiding the development of robust mental models (Robins et al., 2003; Ben-Ari, 2001).

Empirical evidence backs these theoretical views. For example, Scanlan (1987) discovered that most students preferred flowcharts over pseudocode when learning complex algorithms, indicating the natural appeal of visual tools. Recent eye-tracking research shows that students understand flowcharts more quickly and with fewer mistakes than pseudocode, suggesting they help lessen cognitive load during problem-solving (Andrzejewska & Stolińska, 2022). Conversely, some experts argue that pseudocode serves as a crucial mental link between abstract algorithm thinking and real programming languages (Xu et al., 2024). Overall, these studies imply that pseudocode and flowcharts offer distinct benefits: flowcharts boost initial understanding and ease immediate processing, while pseudocode supports the long-term development of coding skills

2.5 The Research Gap: Student Perceptions and Preferences

While the body of literature is rich with studies on the pedagogical effectiveness of flowcharts and pseudocode, there is still a notable research gap concerning students' own perceptions and preferences. Much of the existing research emphasizes performance-based outcomes, such as exam results, programming accuracy, or code correctness, rather than capturing learners' subjective experiences (Gao et al., 2025). For example, Gao et al. (2025) highlight that research in computing education often privileges objective measures over affective dimensions such as motivation, confidence, and perceived usefulness. This leaves open questions about how novice learners personally evaluate and experience algorithm representation tools.

Recent work has begun to address this issue by examining student perceptions in programming education. Keuning et al. (2024) explored how students perceive scaffolding strategies in computing courses and found that students' preferences strongly influenced their engagement and persistence. Similarly, Karnalim and Ayub (2017) investigated the use of a program visualization tool in an introductory programming course and found that students perceived the tool as effective and helpful, highlighting that subjective impressions like enjoyment, perceived ease of use, and satisfaction can contribute significantly to learning satisfaction, independent of measurable performance outcomes. These findings underscore the importance of understanding affective and experiential factors in addition to cognitive outcomes.

Moreover, new studies in 2024 and 2025 show that learners' preferences often diverge from what instructors assume. For example, Yu and Bozic (2023) showed that learners often benefited more from pseudocode when transferring to programming tasks, whereas Chinofunga et al. (2024) reported that instructors tended to emphasize flowcharts as a preferred preparatory tool. Together, these findings highlight a divergence between students' actual preferences and instructors' assumptions about effective introductory representations. Andrzejewska and Stolińska (2022) demonstrated that students' comprehension and confidence varied depending on whether algorithms were presented in flowcharts or pseudocode, highlighting that learners may favor different representations. This suggests that instructional design should be attentive to diverse learning preferences, such as visual and verbal modes.

Taken together, these recent findings reinforce the importance of focusing specifically on student perceptions and preferences. By exploring how first-year computer science students personally assess the effectiveness and usability of pseudocode and flowcharts, this study seeks to fill this gap by directly investigating their subjective perceptions. Through gathering data on their perceived effectiveness and preferences, the research aims to deepen understanding of how these essential tools are received by the learners they are meant to assist. Beyond just measuring performance, this study will provide practical advice for educators on how to effectively integrate these methods to cater to diverse learning needs and foster robust algorithmic thinking from the very start of a student's programming journey.

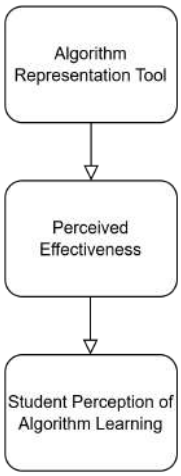

Figure 1: Conceptual Framework Linking Tool Type, Effectiveness, and Perception

Previous studies have focused on performance metrics rather than perceptions, leaving it unclear how subjective evaluation affects algorithm learning. This study fills that gap by empirically linking tool preference, effectiveness and perceptions. Figure 1 above shows a framework that reflects this hypothesized relationship between the three elements, forming the conceptual basis of this study.

3. Methodology

A total of 95 first-year Diploma in Computer Science students enrolled in the Introduction to Algorithm Design and Development course participated in the study. All participants were instructed to use two algorithm representation tools: flowcharts and pseudocode, to ensure direct exposure prior to data collection.

Data were gathered through a structured questionnaire comprising two sections. Section A captured respondents' demographic profile. The second section contained Likert-scale items (1 = Strongly Disagree to 5 = Strongly Agree) designed to measure students' perceptions and perceived effectiveness of flowcharts and pseudocode.

Data analysis was conducted using SPSS version 25. Descriptive statistics were used to summarize respondents' demographic profiles. Since every student experienced both algorithm representation tools, an independent sample t-test was employed to compare the mean perception scores according to gender and prior formal programming course experience before entering UiTM. In addition, One-way ANOVA was used to assess the difference in students' perceptions toward preferred tools in learning algorithm development and the highest level of prior programming experience, while regression analysis was performed to examine the influence of these instructional tools on students' overall perceptions.

4. Results

4.1 Demographic Profile

Table 1
Demographic Profile

Variables		*Frequencies*	*Percentage*
Gender	Male	50	52.6
	Female	45	47.4
Highest level of prior programming experience	No prior experience	6	6.3
	Basic understanding	66	69.5
	Intermediate knowledge	23	24.2
Formal programming course experience before entering UiTM	Yes	31	32.6
	No	64	67.4
Programming Languages	C++	87	91.6
	Java	16	16.8
	Phyton	18	18.9
	PHP	10	10.5
	HTML	3	3.2
	JavaScript	1	1.1
	C#	1	1.1
	HTML & CSS	1	1.1
Preferred Tools	Both	2	2.1

Flowchart	44	46.3
Pseudocode	49	51.6

Table 1 tabulates the frequency and percentage of the respondents' demographic profiles. Most of the respondents were male students (52.6%), while female students accounted for 47.4%. In terms of prior programming experience, 66 students (69.5%) reported having a basic understanding as their highest level, followed by 23 students with intermediate knowledge, and 6 students with no prior programming experience. Most respondents had never taken a formal programming course before enrolling at UiTM, with only 32.6% having attended one. Additionally, C++ was the most used programming language for 87 respondents (91.6%), and more than half of the respondents (51.6%) preferred to use Pseudocode as their tool in learning algorithm development.

4.2 **Reliability Analysis**

Table 2
Results of Reliability Analysis

Variable	***Number of Items***	***Cronbach's Alpha***
Perception with Algorithm Design	8	*0.929*
Perception with Flowcharts	10	*0.943*
Perception with Pseudocode	10	*0.942*
Effectiveness in Learning Algorithm (using flowchart)	8	*0.937*
Effectiveness in Learning Algorithm (using Pseudocode)	8	*0.960*

A reliability test was conducted to assess the consistency of the instruments used in this study, with Cronbach's Alpha being the most applied measure. According to Moidunny (2009), a Cronbach's Alpha value greater than 0.7 indicates that the items are highly reliable. The results presented in Table 2 show that all the items used in this study meet this criterion, demonstrating that the instruments provide consistent and stable results and are therefore suitable for use.

4.3 **Normality Test**

Table 3
Summary of Normality Test

Variable	**Skewness**
Perception with Algorithm Design	-0.601
Perception with Flowcharts	-0.591
Perception with Pseudocode	-0.305
Effectiveness in Learning Algorithm (using flowchart)	-0.838
Effectiveness in Learning Algorithm (using Pseudocode)	-0.406

A parametric statistical test was employed to answer the objectives, provided that the normality assumption is met. A skewness measure is one of the methods that can be performed to check the normality of the data. Since all skewness values shown in Table 3 fall within the range of -1 to 1, the data can be considered normally distributed. Therefore, it is appropriate to use a parametric statistical test for data analysis.

4.4 **Assessing the differences of students' perceptions in learning algorithm development according to gender and formal programming courses before entering UiTM**

Table 4
Means of students' perception by gender, formal programming courses before entering UiTM, and results of the independent t-test

		N	Mean	Levene's Test (Equality of Variances)	Independent t-test (Equality of Means)	Conclusion
Gender	Male	50	3.9943	0.894	0.225	No significant difference
	Female	45	4.1397			
Formal programming course before entering UiTM	No	64	3.9821	0.464	0.050	There is a significant difference
	Yes	31	4.2304			

Table 4 presents the means and results of the independent t-test analyzing students' perceptions based on gender and prior experience in formal programming courses before entering UiTM. To assess the significance of differences between gender groups (male and female) and experience groups (with or without a formal programming course), Levene's test for equality of variances was first conducted. The results in Table 4 indicate that the assumption of equal variances is met for both variables, as the p-values are greater than 0.05.

The Independent t-test results show no significant difference in students' perceptions between males and females (p-value > 0.05). In contrast, there is a significant difference in perceptions between students who have taken a formal programming course before entering UiTM and those who have not (p-value < 0.05). This suggests that gender does not influence perceptions of learning algorithm development, whereas prior formal programming experience significantly affects students' perceptions.

4.5 Comparing students' perceptions toward preferred tools in learning algorithm development and the highest level of prior programming experience

Table 5
Means of students' perception by preferred tools in learning algorithm development, and the highest level of prior programming experience

Preferred tools	N	Mean
Both	2	4.2857
Flowcharts	44	4.1071
Pseudocode	49	4.0146

Highest level of prior programming experience	N	Mean
No prior experience	6	4.2857
Basic understanding	66	3.9827
Intermediate knowledge	23	4.2360

Table 5 summarizes the mean scores of students' perceptions based on their preferred tools for learning algorithm development and their highest level of prior programming experience. The results indicate only a slight difference in perceptions across these two factors, suggesting that students' perceptions of learning algorithm development remain similar. regardless of their preferred instructional tools or prior programming experience.

Table 6
Results of One-Way ANOVA

Group	F-Test	p-value	Conclusion
Preferred tools	0.440	0.646	No significant difference
Highest level of prior programming experience	2.147	0.123	No significant difference

In addition to comparing the mean scores, a further analysis was conducted using one-way ANOVA. This analysis was performed to examine differences in students' perceptions of learning algorithm development based on their preferred tools and highest level of prior programming experience. The results showed no significant differences across these groups, which is consistent with the findings from the mean score comparisons. Despite these preferences, studies indicate that student perceptions and confidence in algorithm learning remain consistent whether using flowcharts or pseudocode, independent of programming experience.

4.6 To explore the effectiveness of preferred tools on students' perception in learning algorithm development

Regression analysis was performed to determine the effect of students' preferred tools on their perceptions and to assess the effectiveness of these tools in influencing their views.

Table 7
Results of regression analysis (Flowchart)

	Coefficient	t	p-value	R^2	F Test
Constant	0.758	3.372	0.001	0.699	215.478
Effectiveness	0.816	14.679	0.000		(0.000)

Table 7 presents the findings on the effectiveness of using flowcharts as a preferred tool among first-year computer science students and its impact on their perceptions. The result indicates that the effectiveness of using flowcharts has a significant influence on students' perception of learning algorithm development (p-value < 0.05). This means there is strong evidence that using flowcharts as a tool positively affects how students view the process of learning algorithms. Approximately 69.9% of the total variation in students' perception can be explained by the effectiveness of using flowcharts, while another 30.1% of the total variation is influenced by other factors. The high percentage of R_2 reveals that flowcharts play an important role in influencing students' perception towards learning algorithm development by providing a clear graphical representation of the steps involved in algorithm development. This result aligns with prior research by Threekunprapa and Yasri (2020), who concluded that the integration of unplugged coding activities with flowcharts enhanced students' understanding of computer science concepts, algorithmic design, and computational thinking.

Table 8
Results of regression analysis (Pseudocode)

	Coefficient	t	p-value	R^2	F Test
Constant	0.978	3.962	0.000	0.629	157.4878
Effectiveness	0.749	12.565	0.000		(0.000)

A similar conclusion can be drawn regarding the effectiveness of using pseudocode as a preferred tool among students in their perceptions (p-value < 0.05). The result reveals that pseudocode is considered an effective tool in the students' perception of learning algorithm development. However, the effectiveness of pseudocode accounts for a smaller proportion compared to flowcharts, explaining 62.9% of the total variation in students' perceptions. It highlights flowcharts as more influential than pseudocode in shaping students' perceptions of learning algorithm development.

5. Discussion

This study examined the comparative perceptions of first-year Computer Science students regarding pseudocode and flowcharts as tools for algorithm development, as well as the perceived effectiveness of these tools in supporting students' understanding of computational problem-solving. The finding of the first objective revealed that there were no significant differences in perception across preferred tools, suggesting that both pseudocode and flowcharts are equally accepted by students. This finding aligns with prior evidence that both visual and textual algorithm representations foster similar levels of comprehension among novice programmers (Grawemeyer et al., 2022; Xu et al., 2024). It reflects a similar perception among first-year computer science students learning algorithm development, regardless of whether they use pseudocode or flowcharts.

Moreover, the significant influence of both tools on perception indicates that visual and textual formats provide complementary cognitive pathways for learning, echoing findings by

Andrzejewska and Stolińska (2022) that visual representations reduce cognitive load and by Ben-Ari (2001) and Vygotsky (1978) who emphasised the value of scaffolding representations in constructivist learning contexts.

Despite its contributions, this study has limitations. Firstly, the research was carried out at a single institution and involved only first-year diploma students, which may restrict the generalizability of the findings to broader groups such as undergraduates in different settings, students from diverse cultural or educational backgrounds, or learners at more advanced stages of programming. Secondly, the study mainly depended on self-reported perceptions collected via questionnaires, which, although useful for understanding subjective experiences, may be affected by response bias or fail to fully reflect the subtleties of students' cognitive processes during algorithm development. Additionally, the cross-sectional approach offered only a snapshot of student perceptions at a single point in time, without considering how preferences and views might change as students gain more programming experience.

Future research should address these limitations by using more diverse and longitudinal designs, including multiple institutions, varied student populations, and repeated measures over time to track the development of student perceptions. Adding qualitative methods like interviews, think-aloud protocols, or classroom observations could deepen understanding, providing further insights into how learners engage with algorithm representation tools in practice. Additionally, future studies could expand the scope beyond flowcharts and pseudocode to incorporate newer pedagogical technologies such as block-based coding environments like Scratch and Blockly, as well as dynamic algorithm visualization systems or AI-powered tutoring tools. Exploring these methods may uncover how modern instructional supports interact with traditional representations to influence not only perceptions but also long-term programming skills, retention, and career preparedness.

6. Conclusion

This study examined first-year computer science students' perceptions and preferences regarding the use of flowcharts, pseudocode, or a combination of both in algorithm development. All students were introduced to both algorithm representation tools, allowing for a direct comparison of how they valued each tool. The findings revealed no significant difference in overall perception scores between these widely adopted instructional methods, indicating that students regarded both flowcharts and pseudocode as useful supports for understanding algorithms. Additionally, the effectiveness of each tool was found to significantly influence students' positive perceptions, underscoring the importance of perceived utility in engaging novice programmers. In conclusion, both flowcharts and pseudocode serve as valuable representations for teaching algorithm development. Unlike previous studies that focused primarily on performance efficiency or cognitive outcomes, this study contributes by emphasizing students' perceived acceptance and learning preferences, offering new insights into how algorithm representation tools can be effectively integrated into early programming education. Teachers are encouraged to use both methods to support diverse learning styles. For future research, it is recommended to explore other tools, such as block-based programming, algorithm animation, or interactive visualization software, and to study their long-term effects on student performance, engagement, and programming retention. Moreover, future investigations could expand on this study by examining the pedagogical impact of integrating multiple representations within a single curriculum, thereby deepening understanding of adaptive teaching strategies in computing education. Additionally, to ensure instrument validity, future studies may employ pilot testing or expert validation and assess construct reliability using Cronbach's Alpha. Such investigations can deepen insights into optimizing pedagogical approaches and technology integration in computing education.

Acknowledgments
The authors would like to acknowledge the support of first-year Diploma in Computer Science (CDCS110) students enrolled in the Introduction to Algorithm Design and Development (CSC121) course at Universiti Teknologi MARA Sarawak, Kampus Samarahan 2, who participated as respondents during the semester from October 2024 - February 2025 and March - August 2025.

Funding Details
This work was not supported by any external funding.

Authors Contributions
Satria Arjuna played a key role in shaping the project's direction by contributing to its initial conceptualization. Zubaidah guided students in applying pseudocode and flowcharts for algorithm development, while Abdul Hadi led the design of the research questionnaire. Both Zubaidah and Abdul Hadi also worked together on drafting and revising the manuscript. Rumaizah contributed her expertise in data analysis, applying statistical and computational methods to interpret the findings. Collectively, their efforts highlight a well-rounded collaboration, with each member bringing distinct skills and shared responsibilities that ensured the success of the research.

Conflict of Interest
The authors declared that they have no conflicts of interest to disclose.

References

Al-Fedaghi, S. (2025). Thinking Machines for Requirements Engineering: Superseding Flowchart-Based Modelling. *arXiv*. https://doi.org/10.48550/arXiv.2501.16712

Andrzejewska, M., & Stolińska, A. (2022). Do structured flowcharts outperform pseudocode? Evidence from eye movements. *IEEE Access, 10*, 132965–132975. https://doi.org/10.1109/ACCESS.2022.3230981

Ben-Ari, M. (2001). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching, 20*(1), 45–73.

Bruner, J. S. (1961). The act of discovery. *Harvard Educational Review, 31*(1), 21–32.

Chinofunga, M. D., Chigeza, P., & Taylor, S. (2024). How can procedural flowcharts support the development of mathematics problem-solving skills? *Mathematics Education Research Journal, 37*, 85–123. https://doi.org/10.1007/s13394-024-00483-3

Clark, J. M., & Paivio, A. (1991). Dual coding theory and education. *Educational Psychology Review, 3*(3), 149–210. https://doi.org/10.1007/BF01320076

Gao, Z., Yan, H., Liu, J., Zhang, X., Lin, Y., Zhang, Y., ... & Feng, J. (2025). Tracing distinct learning trajectories in introductory programming course: a sequence analysis of score, engagement, and code metrics for novice computer science vs. math cohorts. *International Journal of STEM Education*, *12*(1), 27. https://doi.org/10.1186/s40594-025-00546-2

Grawemeyer, B., Halloran, J., England, M., & Croft, D. (2022). Feedback and engagement on an introductory programming module [Preprint]. *Computing Education Practice 2022 (CEP 2022)*. https://doi.org/10.48550/arXiv.2201.01240

Karnalim, O., & Ayub, M. (2017). The effectiveness of a program visualization tool on introductory programming: A case study with PythonTutor. *CommIT (Communication & Information Technology) Journal*, *11*(2), 67–76.

Keuning, H., Alpizar-Chacon, I., Lykourentzou, I., Beehler, L., Köppe, C., de Jong, I., & Sosnovsky, S. (2024). Students' perceptions and use of generative AI tools for programming across different computing courses. *arXiv*. https://doi.org/10.48550/arXiv.2410.06865

Kurniawan, O., Jégourel, C., Lee, N. T. S., De Mari, M., & Poskitt, C. M. (2021). Steps before syntax: Helping novice programmers solve problems using the PCDIT framework. *arXiv preprint arXiv:2109.08896*. https://doi.org/10.24251/HICSS.2022.121

Liu, J., Poulsen, S., Goodwin, E., Chen, H., Williams, G., Gertner, Y., & Franklin, D. (2024). Teaching algorithm design: A literature review. *arXiv*. https://doi.org/10.48550/arXiv.2405.00832

Mayer, R. E. (2004). Should there be a three-strikes rule against pure discovery learning? The case for guided methods of instruction. *American Psychologist, 59*(1), 14–19. https://doi.org/10.1037/0003-066X.59.1.14

Mayer, R. E. (2009). Multimedia learning *(2nd ed.)*. *Cambridge University Press.*

Moidunny, K. (2009). The effectiveness of the national professional qualification for educational leaders (NPQEL). *[Unpublished Doctoral Dissertation, The National University of Malaysia]*, 1-789.

Paivio, A. (1990). Mental representations: A dual coding approach. *Oxford University Press.*

Richter, G., Pister, A., Fekete, J.-D., Sedlmair, M., & Weiskopf, D. (2022). Scalability in visualization. *arXiv*. https://doi.org/10.48550/arXiv.2210.06562

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*(2), 137–172. https://doi.org/10.1076/csed.13.2.137.14200

Sagala, A. A. H., & Yahfizham, Y. (2024). Analisis Pengenalan Konsep Algoritma Pemrograman Matematika Pada Kehidupan Sehari Hari. *Morfologi: Jurnal Ilmu Pendidikan, Bahasa, Sastra dan Budaya*, 2(1), 01-16.

Scanlan, D. A. (1987). Structured flowcharts outperform pseudocode: An experimental comparison. *IEEE Software, 4*(3), 28–36. https://doi.org/10.1109/52.35587

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science, 12*(2), 257–285. https://doi.org/10.1207/s15516709cog1202_4

Tarsini, I. and Anggraeni, R. (2024) Explore flowchart and pseudocode concepts in algorithms and programming, *Indonesian Journal of Multidisciplinary Science*. https://doi.org/10.55324/ijoms.v3i5.807.

Threekunprapa, A., & Yasri, P. (2020). Patterns of computational thinking development while solving unplugged coding activities, coupled with the 3S approach for self-directed learning. *European Journal of Educational Research, 9*(3), 1025-1045. https://doi.org/10.12973/eu-jer.9.3.1025

Vygotsky, L. S. (1978). Mind in society: The development of higher psychological processes. *Harvard University Press.*

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Xu, Z., Zhang, K., & Sheng, V. S. (2024). Logic error localization in student programming assignments using pseudocode and graph neural networks. *arXiv*. https://doi.org/10.48550/arXiv.2410.21282

Yu, J., & Bozic, M**.** (2023, November 9). Investigating the role of pseudocode in learning programming language: A language transfer and typological similarity perspective [Poster]. *Cambridge Language Sciences Annual Symposium*. https://doi.org/10.33774/coe-2023-jdvw9

Zhou, R., Xie, C., He, X., Li, Y., Fan, Q., Yu, Y., & Yan, Z. (2024). Effect of different flow design approaches on undergraduates' computational thinking during pair programming. *Journal of Educational Computing Research*. https://doi.org/10.1177/07356331241268474

UNIVERSITI TEKNOLOGI MARA

eISSN: : 2682-7840