INTERNATIONAL JOURNAL OF ACADEMIC RESEARCH IN PROGRESSIVE EDUCATION & DEVELOPMENT

OPEN ACCESS JOURNAL

Vol 13, Issue 3, (2024) E-ISSN: 2226-6348

Mirror, Mirror, on The Wall, How Do We Measure What Our Students do in Scratch Programming?

Goh Kok Ming[1], Anuthra Sirisena[2], Dayang Rafidah Syariff M. Fuad[3], Juharyanto Juharyanto[4]

Sultan Idris Education University[1,3], SMJK Chung Hwa[2], State University of Malang[4]

Email: kokming888@gmail.com[1], anuthra81@gmail.com[2], dgrafidah@gmail.com[3], Juharyanto.fip@um.ac.id[4]

Corresponding Author Email: kokming888@gmail.com

To Link this Article: http://dx.doi.org/10.6007/IJARPED/v13-i3/21678 DOI:10.6007/IJARPED/v13-i3/21678

Published Online: 15 June 2024

Abstract

This study presents an evaluation framework that aims to enhance the assessment of computational thinking (CT) skills and coding proficiency in students using Scratch programming. The framework takes inspiration from previous research and goes beyond traditional block-counting methods. Instead, it thoroughly analyzes Scratch projects, examining the types of blocks used and how they interact within the program. By incorporating established CT concepts and indicators of project complexity, the framework provides educators with a comprehensive approach to evaluating student projects. While implementing the framework may pose challenges in terms of manual assessment and scalability, it holds promise in fostering the development of vital CT skills in students and preparing them for success in an increasingly digital world. Further refinement and validation of the framework are necessary to ensure its effectiveness and applicability in diverse educational settings.

Introduction

Over the past decade, there has been a rise in the integration of programming and computational thinking (CT) skills in educational curricula (Chen et al., 2023). CT, first introduced by Wing (2006), emphasizes problem-solving methods rooted in computer science principles. This emphasis has grown in response to our technology-driven society. Although coding classes have been available since the 1970s and 1980s, they were often difficult to access due to the complexity of traditional programming languages. These languages require students to grapple with intricate symbols and syntax, creating a significant obstacle for younger learners. Fortunately, the emergence of new visual programming languages has brought about a new era. Languages like Scratch and Alice provide user-friendly interfaces that allow young students to grasp fundamental programming concepts without getting caught up in intricate syntax details (Chang, 2014). This creates an engaging and accessible learning experience that sparks a passion for technology and problem-solving in a new generation (Resnick, 2009).

Since its introduction in 2009, Scratch has become a popular platform for young programmers aged six and above. It offers an accessible space for creating and sharing programs using intuitive building blocks (Resnick et al., 2009). Scratch has gained millions of users worldwide and is known for its interactive approach, captivating learners of all ages. Aside from its entertainment value, Scratch is a powerful tool for developing problem-solving and critical-thinking skills. It has been proven to enhance comprehension in various subjects (Kalelioglu & Gulbahar, 2014; Piedade & Dorotea, 2022). Educational institutions, from primary schools to universities and after-school programs, have embraced Scratch as a way to introduce students to the exciting world of coding. Moreno-León et al (2018) propose a shift from generic programming to computational thinking (CT). They emphasize the importance of nurturing problem-solving abilities and proficiency in tackling complex issues through CT techniques such as decomposition and algorithm design. This approach equips students with structured and logical problem-solving methods while fostering creativity and collaboration. It prepares individuals for success in today's digital landscape (Berikan & Özdemir, 2019; Korkmaz, 2016).

However, educators face a significant challenge when it comes to assessing students' learning. One major issue is the lack of effective tools for evaluating student programs and assessing their development of CT skills through Scratch (Moreno-León et al., 2015). This challenge is partly due to the absence of a universally agreed-upon definition of CT, which leaves educators unsure about the best methods for integrating it into their curriculum (Grover & Pea, 2013; Li et al., 2020a). The different operational definitions of CT lead to a diversity of assessment practices (Stewart & Baek, 2023). This study is critical for several reasons. First, it addresses the growing need to equip students with essential CT skills, which are increasingly vital for success in various professional fields and everyday problem-solving. The integration of CT into educational curricula helps develop students' logical thinking, creativity, and collaboration abilities, preparing them for a digital future. Moreover, understanding the effectiveness of tools like Scratch in fostering these skills can guide educators in implementing more effective teaching strategies. Therefore, the research questions for this study are as follows: (i) What existing evaluation models or frameworks for Scratch projects can assist teachers in assessing students' projects? and (ii) What elements of Computational Thinking are utilized in the evaluation of Scratch projects?

Literature Review

Scratch Programming

Scratch is a visual programming language primarily designed for education. It uses a block-based system, allowing users to create applications by combining visual components such as images, sounds, and videos with scripted functionalities. The logic of these applications is defined by assembling pre-defined blocks, similar to building with Lego bricks (Ford (Jr.), 2014; Stewart & Baek, 2023). Each block represents a specific command or action, guiding the application through various tasks. Additionally, Scratch provides a wide range of media resources, including graphics and sound effects, as well as features for designing personalized graphical and audio elements (Ford (Jr.), 2014). It offers a comprehensive environment that empowers users of all ages (Stewart & Baek, 2023). The intuitive visual programming language removes barriers, allowing beginners to easily dive into application development and receive immediate feedback on their creations (Ford (Jr.), 2014). This fosters a hands-on learning experience that helps users grasp fundamental programming concepts (Rose et al., 2017).

Beyond being a programming language, Scratch is also a valuable pedagogical tool when integrated into different subject areas and learning environments (Stewart & Baek, 2023; Voinohovska & Doncheva, 2021; Resnick & Rusk, 2020). According to the study by Silva et al. (2022), integrating Scratch into mathematics education enhances the teaching and learning process. This approach promotes a more meaningful, creative, and playful learning experience, as reported by four Mathematics undergraduate students from a Federal Public Institution in the Midwest. In a pilot project conducted by Naz et al (2017), the effectiveness of Scratch programming in K-12 classrooms was investigated. The project aimed to support nineteen K-12 teachers from middle and high schools with no prior programming experience. The teachers received training in a "hybrid format" that combined face-to-face and online training. Pre- and post-assessments revealed a significant improvement in students' performance when utilizing Scratch programming. Furthermore, Bahar's (2021) study explored the integration of Scratch, a visual programming language, into language teaching for children. The study aimed to assess its impact on language development and cognitive skills, as well as identify benefits and challenges from the perspectives of students and teachers. The findings revealed a significant positive impact on children's listening and computational thinking skills. While there was no statistically significant effect on academic achievement, language use, or narrative skills, the experimental group consistently outperformed the control group in nine out of thirteen components. Students responded positively to Scratch, finding enjoyment in the lessons, project creation, and collaborative work. Overall, the study suggests that the benefits of Scratch in language learning outweigh the identified challenges.

In terms of the learning environment, Presicce et al (2020) conducted a series of online workshops called "WeScratch" which aimed to assist educators in creating engaging online learning environments that foster creativity. WeScratch provides a welcoming and playful space where educators can actively learn to code using Scratch while experiencing an alternative approach to online learning. This study aimed to examine how WeScratch promotes engagement among educators worldwide. By providing a playful and collaborative environment, WeScratch encouraged educators to experiment and enhance their Scratch programming skills. Moreover, Presicce et al (2020) provided examples of how educators value their WeScratch experience, both in terms of personal skills development and as a model for designing similar learning experiences for their students.

Therefore, previous research highlights Scratch as a versatile tool that can empower learners of all ages. Its user-friendly interface and engaging nature make it suitable for use in various educational contexts, from fostering computational thinking skills in primary education classrooms to promoting creative expression in language learning. Educators can leverage Scratch's rich media library and block-based programming system to design interactive lessons that cater to different learning styles and abilities. Furthermore, Scratch's online community provides a platform for learners to share their creations, collaborate on projects, and engage in peer-to-peer learning, fostering a sense of community and promoting deeper engagement in the learning process.

Challenges of Scratch Programming

Assessing computational thinking (CT) in Scratch Programming presents challenges due to the existence of different definitions of CT (Stewart & Baek, 2023; Moreno-León et al., 2016). In a meta-analysis conducted by Sun et al. (2021), various methods of assessing CT skills in K-12 students during programming activities were explored. The study identified four main forms of CT assessment: CT tests, CT scales, CT tasks, and CT questions. Recent studies indicate that CT scales, CT tests, and CT tasks are preferred as assessment tools, while CT questions are less common due to their time-consuming nature and potential influence on participants (Sun et al., 2021).

Zhao et al (2022) provided examples of these four forms of CT assessment mentioned by (Sun et al., 2021). CT tests such as Bebras Dagiene & Stupuriene (2016) and CTT Roman-González et al (2018) are widely used to assess learners' ability to apply CT skills in different scenarios or challenges. CT scales such as CTS Korkmaz et al (2017) are commonly used for comprehensive evaluation of learners' CT proficiency. CT tasks such as Dr. Scratch Ma et al (2021) are utilized as formative assessment methods to evaluate learners' CT skills. On the other hand, CT questions involve researchers verbally presenting inquiries to the subjects. As noted by Stewart & Baek (2023); Moreno-León et al (2016), the diverse interpretations of CT lead to varied assessment practices, which can pose two main issues. Firstly, it becomes challenging to compare student achievement across different educational programs or institutions. Secondly, there can be confusion regarding the essential CT skills and knowledge that students should acquire.

Furthermore, Orozco-Garcia et al (2019) stated that while existing evaluation tools allow for the assessment of CT development, they do not provide teachers with the ability to (i) select the factors to be evaluated, (ii) determine the complexity level of the tasks, or (iii) provide continuous follow-up for timely feedback to students regarding their performance. Therefore, Orozco-Garcia et al (2019) developed a formative assessment tool for Scratch programming. The web tool, HERA, enables teachers to design challenges for their students, assess CT dimensions, provide feedback through challenge results and a gamified strategy, and track each student's CT development. In terms of coding behavior, Lye and Koh (2018) examined three case studies to investigate how elementary school children, with different levels of programming abilities, approach Scratch programming. Lye and Koh (2018) propose instructional implications to enhance support for children's engagement in computational thinking during K-12 programming lessons.

Based on previous literature, some assessments heavily focus on coding ability, while others emphasize problem-solving strategies or algorithmic thinking. This inconsistency makes it difficult to compare student progress across different programs or even within the same curriculum. To address this, there is a need to clearly define the specific CT concepts and practices being assessed within a particular context. This allows teachers to design targeted assessments that accurately measure student learning and identify areas where additional support may be needed.

Methodology

This study adapted Wohlin's (2014) guidelines for conducting a systematic literature review with a snowballing approach. A search was conducted in Google Scholar to identify the related literature with the search string with keywords: “Scratch assessment”, “Scratch evaluation” and “Scratch programming framework”. The search used the time interval 2010-2024. The criteria and process for selecting the literature are presented in Table 1 and Figure 1 to prepare a start set for the next step. Once a start set was identified, snowballing techniques were performed to determine the inclusion of articles. An analysis was performed to address the research questions based on the findings.

Table 1
Inclusion and exclusion criteria

Aspect	Inclusion Criteria	Exclusion Criteria
Time interval (Year)	Articles in the time interval from 2010 till 2024	Articles are not in time interval from 2010 till 2024
Types of publication	Full papers or articles are included	Conference papers, proceedings, concept papers, review papers, and book chapters are excluded
Focus	Articles that focus on the keywords: “Scratch assessment”, “Scratch evaluation” and “Scratch programming framework” are included.	Articles that do not focus on the keywords: “Scratch assessment”, “Scratch evaluation” and “Scratch programming framework” are excluded.
Language	English	Non-English

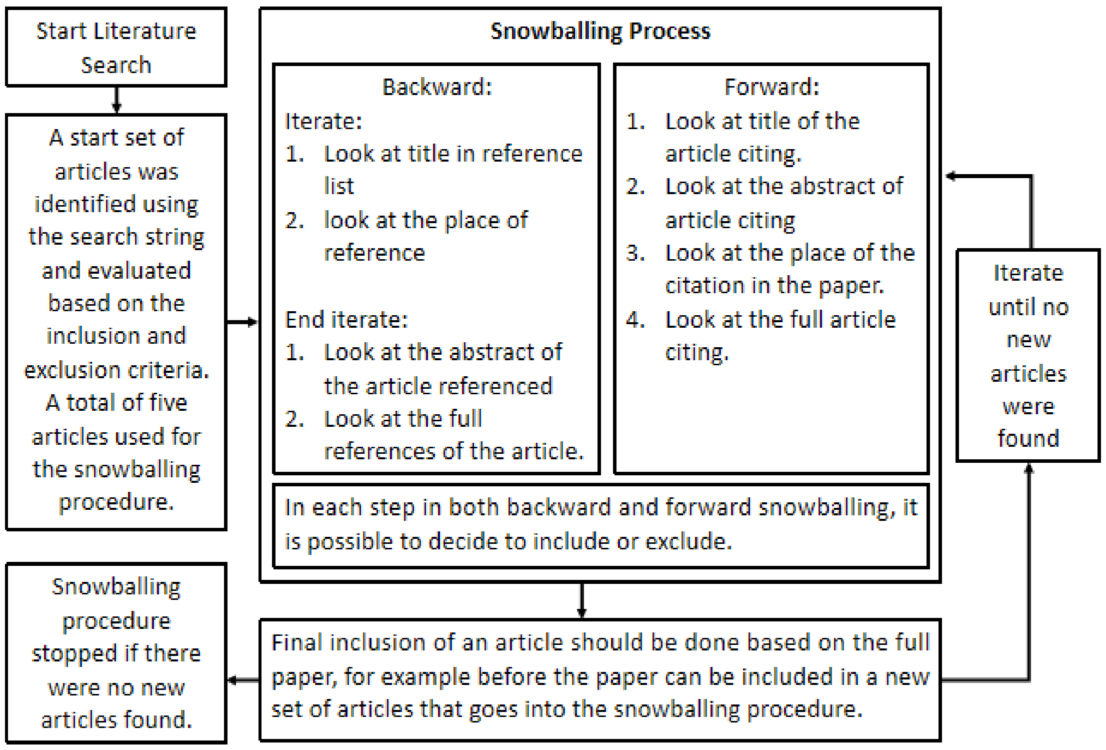
Start Literature Search
A start set of articles was identified using the search string and evaluated based on the inclusion and exclusion criteria. A total of five articles used for the snowballing procedure.
Snowballing Process
Backward:
Iterate:
1. Look at title in reference list
2. look at the place of reference
End iterate:
1. Look at the abstract of the article referenced
2. Look at the full references of the article.
Forward:
1. Look at title of the article citing.
2. Look at the abstract of article citing
3. Look at the place of the citation in the paper.
4. Look at the full article citing.
In each step in both backward and forward snowballing, it is possible to decide to include or exclude.
Iterate until no new articles were found
Final inclusion of an article should be done based on the full paper, for example before the paper can be included in a new set of articles that goes into the snowballing procedure.
Snowballing procedure stopped if there were no new articles found.

Figure 1. The research approach of this study.

Threats to Validity
This study acknowledges potential biases in searching the relevant literature. Selection bias could have arisen from choosing related articles based on the snowballing search strategy.

However, the selection was based on pre-defined content criteria (inclusion and exclusion criteria). Topic relevance was secondary, as the focus was on search strategy assessment. Assessor bias was minimized by involving three researchers who independently assessed the articles' quality. This concept, along with the introduction of "borderline articles", enhances transparency in inclusion/exclusion decisions, allowing readers to assess potential bias. Overall, the study design and pre-defined article selection criteria effectively minimize threats to validity.

Findings

Existing Evaluation Framework for Scratch Projects

When working with the Scratch programming language, researchers used different methods for assessing the development of computational thinking (CT) among learners (Moreno-León et al., 2016). Some suggest using quantitative measures like test scores and completion rates to track CT progression in Scratch users. Others recommend qualitative approaches such as interviews and observations to gain a deeper understanding of learners' problem-solving abilities and creativity (Stewart & Baek, 2023). A study by Wilson et al (2012) addressed a gap in research on the effectiveness of game-based learning (GBL) and game-based construction (GBC) in primary education. While these methodologies are gaining popularity for enhancing student engagement and motivation, especially in teaching computer programming, there is still a lack of empirical evidence supporting their effectiveness. Wilson et al (2012) analyzed 29 games created by students in grades 4 through 7 using Scratch to assess the participants' level of programming proficiency. The analysis was based on predefined criteria designed to evaluate programming skills within a GBC framework. The study offered valuable pedagogical insights and established guidelines for evaluating programming ability in this specific context.

Brennan and Resnick (2012) also examined how design-based learning activities, especially programming interactive tools with Scratch, can enhance computational thinking skills. They proposed a framework consisting of three essential aspects: computational thinking concepts (e.g., iteration), practices (e.g., debugging), and perspectives (e.g., problem-solving approach). They then discussed the use of evolving assessment methods such as project analysis, interviews, and design scenarios. This study further explored the assessment strategies employed by the researchers, investigating how techniques like project portfolio analysis, artifact-based interviews, and design scenarios can be used to evaluate each of these framework dimensions. By dissecting these assessment techniques, Brennan and Resnick (2012) provide valuable insights for educators interested in evaluating the learning that occurs when young people participate in programming activities. Moreover, Seiter and Foreman (2013) introduced the Progression of the Early Computational Thinking (PECT) Model. This model provides a comprehensive approach to assessing computational thinking in primary grades (Grades 1 to 6). The PECT Model combines measurable evidence from student work with coding design patterns aligned with core computational thinking concepts. In a pilot test study, Seiter and Foreman (2013) demonstrated the effectiveness of the PECT Model in identifying variations in computational thinking abilities among students of different ages and tracking overall progress toward increased computational sophistication. These findings are crucial for developing age-appropriate curricula that target computational thinking skills in primary education. By considering students' cognitive development stages, the PECT Model enables educators to design engaging lesson plans that optimize learning experiences for each grade level.

In the study conducted by Moreno-León et al (2015), the researchers recognized a lack of effective tools for assessing student projects. To address this gap, they introduced Dr. Scratch, a web application designed to assist educators in automatically analyzing Scratch projects. Dr. Scratch provides students with immediate feedback on their code, identifying errors, offering suggestions for improvement, and promoting the development of Computational Thinking (CT) skills. To evaluate the application's effectiveness, researchers conducted workshops with students aged 10-14 from eight different schools. More than 100 participants used Dr. Scratch to analyze their Scratch projects, reviewed the feedback provided by the tool and implemented its suggestions for improvement. The findings of the study revealed a significant increase in students' CT scores and coding skills after the workshops, highlighting the potential value of Dr. Scratch as a tool for enhancing formative assessment and student motivation in computer programming education. Building upon the work of Moreno-León et al (2015); Ngeow (2016) proposed a new framework called Scratch School for analyzing Scratch projects. This framework aimed to address the limitations of existing analysis tools by focusing on result stability, data storage, and advanced features. Scratch School utilizes a novel algorithm design to improve analysis accuracy. The proposed framework employs two primary analysis methods: basic analysis for overall project evaluation and question-based analysis. Users can create and submit questions with specific requirements, and receive analysis results based on their answers.

A study conducted by Nančovska Šerbec et al (2018) compared primary school students (aged 8-12) with future computer science teachers. The study analyzed their cognitive processes through Scratch projects. The results showed that both groups had a similar understanding of flow control and data representation. However, students displayed weaker performance in logic, synchronization, and parallelism. These findings suggest that there are differences in reasoning abilities and comprehension of complex or concurrent events between the two groups. The study also emphasized the potential for enhancing elementary students' computational thinking skills through guided game programming activities. Additionally, the complexity level of Scratch projects emerged as a significant factor in evaluating students' computational thinking skills. While computational thinking encompasses problem-solving, algorithmic thinking, and abstraction, the complexity of Scratch projects provides further insights into students' comprehension and application of these concepts. Previous research has primarily used code analysis to assess computational thinking skills in Scratch programming. Although this approach provides feedback on competence, it lacks the contextual richness offered by observations or interviews. Automated and performance-based approaches often fail to offer explicit suggestions or tips for improving code, especially in terms of efficiency or complexity. Therefore, this study highlights the need for more comprehensive evaluation and assessment strategies.

Furthermore, Chai et al (2021) discovered that existing auto-judgment tools, which rely on rigid criteria, were struggling with the complexity of Scratch projects. This poses a challenge for manual project evaluation due to the rich multimedia content and diverse project types in Scratch programming. To address this challenge, Chai et al (2021) proposed a Dynamic Weighted Evaluation System (DWES). The DWES establishes a new evaluation framework based on eight key computational thinking (CT) criteria and automatically assesses student projects against these criteria. DWES dynamically adjusts the evaluation results based on project type, moving away from a one-size-fits-all approach. Therefore, this data-driven adjustment takes into account the specific CT performance and scripts within each project. Inspired by Moreno-León et al (2015); Chai et al (2021); Ang (2023) proposed a dynamic weighted evaluation system (DWES) that evaluates the Scratch projects based on CT concepts and project type. Ang (2023) refined the CT evaluation criteria by identifying eight key concepts: abstraction and problem decomposition, parallelism, logical thinking, synchronization, flow control, user interactivity, data representation, and code organization. The latter modification was taken from the previous work of (Moreno-León et al., 2015). Ang (2023) expanded the scoring system from a maximum of 3 points to 5 points per concept. This revised system assigns points based on different levels of competence, ranging from "Basic" (1 point) to "Proficient" (5 points) (Chai et al., 2021). This change provides a more nuanced assessment of students' understanding of each CT concept within their Scratch projects.

Zhao et al (2022) examined the impact of mind mapping on students' computational thinking (CT) skills when learning programming with Scratch. The adapted CT scale and CT tasks (Dr. Scratch) were used to assess computational thinking abilities in the study. The CT scale was used as a summative assessment tool, while the CT task was a formative assessment tool. Both tools were employed to measure the students' computational thinking abilities. The study involved 73 fifth-grade students (aged 10 to 11) from public primary schools. The findings of the study revealed that the use of mind mapping was effective in improving students' CT skills. The majority of these methods rely on manual analysis at the beginning. Because of this, there was a need for more automated tools to assist in this process, as seen in the work of (Ngeow, 2015; Moreno-León et al., 2015; Chai et al., 2021; Ang, 2023). However, according to a study by Smith et al (2018), automated tools can make it more difficult for learners to receive an accurate assessment of their CT development. Additionally, relying solely on automated tools can overlook important qualitative aspects of learners' programming abilities (Stewart & Baek, 2023). Therefore, we recognize the significance of manual assessment of student progress in developing computational thinking skills through Scratch. Manual assessment allows educators to consider not only quantitative data but also qualitative aspects of students' problem-solving abilities and logical reasoning. For example, when assessing a student's ability to write efficient code, automated tools may only focus on the correctness of the output without considering the elegance or readability of the code. By manually evaluating the student's work, educators can provide feedback on not only the final result but also on their thought process and problem-solving approach. This assessment approach can better support students in developing a well-rounded understanding of computational thinking skills through Scratch. Therefore, educators should prioritize providing feedback that highlights both strengths and areas for improvement in students' problem-solving approaches through manual assessment of Scratch projects.

What elements of Computational Thinking are utilized in the evaluation of Scratch projects?
A total of five (5) articles on the evaluation framework for Scratch programming were found in the search results (Brennan & Resnick, 2012; Moreno-León et al., 2015; Ngeow, 2016; Chai et al., 2021; Ang, 2023). These articles present different components and concepts for assessing the complexity level and computational thinking (CT) competence of students' Scratch projects, whether through automated or manual methods. The literature suggests various approaches to evaluating Scratch projects Moreno-León et al (2016), such as analyzing block types and their interactions within the program, as well as integrating established CT concepts and indicators of project complexity. Although these studies provide valuable

insights into improving the evaluation process, challenges like manual assessment and scalability may impede implementation. Nevertheless, the synthesis of these findings highlights the importance of developing a comprehensive evaluation framework for Scratch projects that can effectively measure students' CT skills and coding proficiency.

Table 2
Components and concepts identified across different articles

Components/Concepts	Brennan and Resnick (2012)	Moreno-León et al. (2015)	Ngeow (2016)	Chai et al. (2021)	Ang (2023)
Computational Thinking (CT)	√	√	√	√	√
Abstraction		√	√	√	√
Problem Decomposition		√		√	√
Parallelism	√	√		√	√
Logic Thinking		√	√	√	√
Conditionals	√				
Operators	√				
Synchronization		√	√	√	√
Flow Control		√		√	√
Sequences	√				
Loops	√				
User Interactivity		√		√	√
Data Representation	√	√		√	√
Code Organization				√	
Events	√				
Algorithms			√		
Patterns			√		
Complexity Level			√		

Drawing upon the components and concepts identified in Table 2 from existing studies, a comprehensive evaluation framework for Scratch projects could be proposed. This framework combines both computational thinking (CT) competence and complexity, aiming to provide a holistic assessment of students' programming endeavors. The components selected for CT competence within this framework encompass crucial aspects such as abstraction and problem decomposition, parallelism, logic thinking, synchronization, flow control, user interactivity, data representation, and code organization. Through this proposed framework, educators can gain insights into students' abilities to conceptualize, design, and implement solutions within the Scratch programming environment. However, further validation and refinement of the framework are necessary to ensure its effectiveness and applicability across diverse educational contexts.

Discussion

The assessment of computational thinking (CT) skills in Scratch programming is a complex task. There are various methods and complexities involved, and numerous assessment tools, such as CT tests, scales, tasks, and questions, have been discussed. However, their effectiveness and applicability are still being tested. Several studies by Moreno-León et al (2015); Chai et al (2021); Ang (2023) have explored automated evaluation systems, recognizing their potential but also noting limitations in fully capturing students' programming abilities. The dynamic weighted evaluation system (DWES), introduced by Chai et al (2021); Ang (2023), shows promise in assessing Scratch projects, but it may not fully consider the qualitative aspects that are crucial for CT skills assessment. Similarly, Zhao et al (2022) have introduced mind mapping to foster CT development, but questions about scalability and applicability arise. Stewart and Baek (2023) emphasize the importance of integrating qualitative dimensions alongside quantitative metrics in manual assessments to gain a comprehensive understanding of students' CT skills. Additionally, project complexity in Scratch projects is often overlooked but plays a vital role in understanding student engagement and proficiency. By integrating assessments of project complexity into CT skills evaluations, teachers can provide more comprehensive support and foster enhanced learning experiences in Scratch programming.

The exploration of evaluation frameworks for Scratch programming, which includes contributions from Brennan and Resnick (2012); Moreno-León et al (2015); Ngeow (2016); Chai et al (2021); Ang (2023), highlights the dynamic nature of assessing students' CT skills. By synthesizing these findings, there is a need to have a comprehensive evaluation framework for Scratch projects is necessary to effectively measure students' CT skills. Building upon identified components and concepts, a proposed framework could be proposed to assess students' CT skills. This synthesis not only provides teachers with valuable insights into students' ability to conceptualize and execute solutions within Scratch but also highlights the ongoing need for validation and refinement to ensure the framework's effectiveness across diverse educational contexts.

Conclusion

In conclusion, the proposed evaluation framework for Scratch projects builds upon previous research to offer a comprehensive assessment of students' coding proficiency and computational thinking (CT) skills within the Malaysian educational context. It goes beyond simply counting blocks by providing educators with deeper insights into how students apply programming concepts and understand coding fundamentals. While potential evaluator bias, scoring system rigidity, and scalability challenges should be considered, the framework represents a significant advancement in educational assessment. It provides educators with a valuable tool to develop essential CT skills in students. Future refinements and validation efforts will ensure its effectiveness and applicability across diverse educational settings. By leveraging the strengths of the framework and addressing its limitations, educators can empower students to navigate the complexities of the digital age and prepare them for success in a tech-driven world.

References

Bahar, N. (2021). The Effect of Scratch on Children's English Language and Cognitive Development. Retrieved on 23 March 2024 from https://open.metu.edu.tr/bitstream/handle/11511/89840/12626190.pdf

Brennan, K., & Resnick, M. (2012). Using artifact-based interviews to study the development of computational thinking in interactive media design. Paper presented at annual American Educational Research Association meeting, Vancouver, BC, Canada.

Chai, X., Sun, Y., Luo, H., & Mohsen Guizani. (2021). DWES: A Dynamic Weighted Evaluation System for Scratch based on Computational Thinking. IEEE Transactions on Emerging Topics in Computing, 1–1. https://doi.org/10.1109/tetc.2020.3044588

Chang, C. K. (2014). Effects of Using Alice and Scratch in an Introductory Programming Course for Corrective Instruction. Journal of Educational Computing Research, 51(2), 185–204. https://doi.org/10.2190/ec.51.2.c

Ford (Jr.), J. L. (2014). Scratch 2.0 Programming for Teens. In Google Books. Cengage Learning PTR.

Kalelioglu, F., & Gulbahar, Y. (2014). The Effects of Teaching Programming via Scratch on Problem Solving Skills: A Discussion from Learners' Perspective. Informatics in Education. 13. 33-50.

Li, Y., Schoenfeld, A. H., diSessa, A. A., Grasser, A. C., Benson, L. C., English, L. D., & Duschl, R. A. (2020a). Computational thinking is more about thinking than computing. Journal for STEM Education Research, 3(1), 1–18. https://doi.org/10.1007/s41979-020-00030-2.

Lye, S. Y., & Koh, J. H. L. (2018). Case Studies of Elementary Children's Engagement in Computational Thinking Through Scratch Programming. Computational Thinking in the STEM Disciplines, 227–251. https://doi.org/10.1007/978-3-319-93566-9_12

Moreno-León, Jesús. (2018). On the Development of Computational Thinking Skills in Schools through Computer Programming with Scratch. 10.13140/RG.2.2.12797.05609.

Naz, A., Cody, M., Zackoski, R., Caleb, M., & Dingus, R. (n.d.). Applying Scratch Programming to Facilitate Teaching in k-12 Classrooms. Retrieved April 8, 2024, from https://peer.asee.org/applying-scratch-programming-to-facilitate-teaching-in-k-12-classrooms.pdf

Orozco-Garcia, L., González, C., Carlos, J., Cristian Mondragón, & Hendrys Tobar-Muñoz. (2019). A Formative Assessment Tool to Support Computational Thinking in the Classroom. https://doi.org/10.1109/icvrv47840.2019.00043

Piedade, J., & Dorotea, N. (2022). Effects of Scratch-based activities on 4th-grade students' computational thinking skills. Informatics in Education. https://doi.org/10.15388/infedu.2023.19

Seiter, L., & Foreman, B. (2013). Modeling the learning progressions of computational thinking of primary grade students. Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research - ICER '13. https://doi.org/10.1145/2493394.2493403

Silva, J. M. P. da, Nogueira, C. A., Pina Neves, R. D. S., & Silva, P. C. B. (2022). A utilização do Scratch como ferramenta pedagógica na percepção de quem ensinará matemática. Revista Brasileira de Ensino de Ciência E Tecnologia, 15(2). https://doi.org/10.3895/rbect.v15n2.9614

Wilson, A., Hainey, T., & Connolly, T. (2012). Evaluation of Computer Games Developed by Primary School Children to Gauge Understanding of Programming Concepts. In P. Felicia

(Ed.), Proceedings of the 6th European Conference on Games Based Learning (pp. 549-558). Academic Conferences and Publishing Limited (ACPIL).

Wohlin, C., Kalinowski, M., Felizardo, R. K., & Mendes, E. (2022). Successful combination of database search and snowballing for identification of primary studies in systematic literature studies. Information and Software Technology, 147(147), 106908. https://doi.org/10.1016/j.infsof.2022.106908

Voinohovska, V., & Doncheva, J. (2021). The Potential Of Scratch As An Educational Environment For Teaching Students With Special Educational Needs. INTED Proceedings (Internet). https://doi.org/10.21125/inted.2021.0021