Jurnal Inovasi Teknologi Pendidikan
Volume 7, No. 2, Oktober 2020 (177-187)
Online: http://journal.uny.ac.id/index.php/jitp

Primary school pre-service teachers competence level of computational concepts in programming using Dr. Scratch

Theresia Yunia Setyawan *
Universitas Sanata Dharma.
Jl. Affandi, Santren, Caturtunggal, Depok, Sleman, Daerah Istimewa Yogyakarta 55281, Indonesia.
theresiayunia@usd.ac.id
***** Corresponding Author

ARTICLE INFO

Article History
Received:
30 November 2020;
Revised:
6 January 2020;
Accepted:
19 January 2020

Keywords
Computational concepts;
Dr. Scratch;
Pre-service teachers;
Primary school

ABSTRACT

This descriptive research aimed at providing a general description of primary school pre-service teachers' competence level of computational concepts in programming with Scratch using Dr. Scratch. This study analyzed Scratch projects made by 87 sophomore students of the Primary School Teacher Education Program of Sanata Dharma University as part of their Media Pembelajaran Berbasis ICT (MPBICT) course. The projects were then submitted to the Scratch Online Community platform and analyzed using a web tool called Dr. Scratch to analyze their competence level of computational concepts. The analysis results provided by Dr. Scratch showed that 75.86% of the Scratch projects belong to the category of developing projects while 22.99% of them were categorized as master projects, and only 1.15% of the projects could be labeled as basic projects. The results also revealed that the most common bad coding practices identified in the submitted projects were duplicated scripts and object namings. These results indicated that the primary school pre-service teachers of the Primary School Teacher Education Program of Sanata Dharma University had moderate competence level in integrating Scratch computational concepts such as flow control, data representation, synchronization, and user interactivity but the pre-service teachers still needed to be provided with more opportunities to work with other Scratch computational concepts such as abstraction, parallelism, and logic.

INTRODUCTION

It is undeniable that we are living in a fast-moving digital world where new technologies emerge almost every day. In order to survive and thrive in a world mostly driven by computers and software, one needs skills to understand what the problems are, skills to tackle them, and therefore, try to develop possible solutions to the problems. One needs to understand what is happening around them by learning how to think and work systematically as computers and software do. This ability to think and solve problems as performed by computers and software has long been known as computational thinking (CT) skills. Though the term contains the word compute in it, CT is not necessarily about computers and computing only. Barr et al. (2011) implied that CT was already manifested in many dispositions or attitudes we can see in our everyday lives. Confidence in dealing with complexity, persistence in working with difficult problems, tolerance for ambiguity, the ability to deal with open-ended problems, and communicating and working with others to achieve a common goal or solution have become essential dimensions of CT seen in everyday life. To put it simply, the concept of CT has already been used to deal with problems across different contexts and aspects of life.

As next-generation leaders, our students need to master the skills of CT as soon as possible because the skills not only build the good kind of attitudes mentioned above but also serve as the foundation of thought and problem solving (McVeigh-Murphy, 2019). CT skills will empower students to be organized in their work by reflecting on the problems they encounter and intentionally developing solutions for them. They will also be more persistent through iteration and experimentation and able to work in a collaborative environment. More importantly, they will be able to make their own inquiry and learn to embrace ambiguity and reframe challenges as opportunities to develop a growth mindset that will hopefully lead to their ability to be lifelong learners (Fingal, 2018).

Considering the importance of CT, many schools and educational institutions have been making their ways to incorporate it across their curricula. As suggested by Angevine (2018), CT skills could be taught across disciplines, so they were not restricted to computer science subjects only. Hunsaker (2020) proposed that a particularly effective way to introduce CT to students was by teaching coding, as it could help them visualize and experience the concepts in a more concrete way. Though CT skills can also be taught in any subject area that does not require coding, it is definitely a fun and obvious way to learn core components of CT, such as decomposition, pattern recognition, abstraction, algorithmic thinking, and evaluation (Vaidyanathan, 2016).

Research has shown that one of the most popular yet effective tools to teach the concepts of CT is Scratch (Fagerlund et al., 2021; Kotsopoulos et al., 2017; Weese & Feldhausen, 2017). Scratch is a free block-based visual programming language developed by the Massachusetts Institute of Technology (MIT) Media Lab. As noted by (Vatansever & Baltaci Goktalay, 2018), Scratch provided students with an environment that encouraged multifaceted thinking as well as creative problem-solving skills, which were significant components of CT. Scratch can engage students with a set of CT concepts, such as performing a sequence of steps, executing multiple sequences through loops, implementing several different sequences at the same time through parallelism, triggering one sequence through events, making decisions based on conditions, performing mathematical operations through operators, and using data and variables (Voinohovska et al., 2019). One of the tools that can assess students' CT skills as reflected in their Scratch projects is Dr. Scratch (Šerbec et al., 2018). Dr. Scratch is a free and open source web application designed to analyze a set of concepts underlying projects programmed with Scratch (Martins-Pacheco. et al., 2019; Román-González et al., 2017). It is a simple analytic tool that provides automatic feedback for educators and learners on using computational concepts in Scratch projects.

This study aimed to analyze pre-service teachers' competence level of computational concepts as reflected in their Scratch projects. The students were sophomore students of the Primary School Teacher Training Program of Sanata Dharma University, taking Media Pembelajaran Berbasis ICT (MPBICT) as one of their compulsory subjects. They were taught and asked to use Scratch to design educational games for elementary school students. Their projects were then analyzed using Dr. Scratch to identify the level of computational concepts they had used in their projects. It is worth noting that identifying these pre-service teachers' competence level of computational thinking will serve as a preliminary step in the attempts to understand the concepts. For primary school teacher candidates, this understanding is essential in helping their future students develop a variety of cognitive skills, such as number sense, language skills, and visual memory, through both programming and non-programming activities they design (Clements in Elskamp, 2018; Jacob & Warschauer, 2018; Rich et al., 2020).

Computational Thinking and Computational Concepts

Loosely defined, computational thinking (CT) is thinking like a computer to solve problems. The term itself dates back to the works of (Papert, 1993), who envisioned computers as teaching machines that might affect the way people thought and learned, enhance thinking, and change human patterns of access to knowledge. Wing (2006) was the first to coin the term computational thinking to articulate a vision that everyone, not just those who major in computer science, could benefit from thinking like a computer scientist. Later on, Wang (2017) defined it as is the mental skill to apply fundamental concepts and reasoning derived from modern digital computers and computer science,

in all areas, including day-to-day activities. In other words, CT can be defined as problem-solving processes needed to address authentic and real-world issues (ISTE & CSTA, 2011).

Through their work on Scratch, Brennan and Resnick (2012) viewed computational thinking as a device for conceptualizing the learning and development that took place within the program. They developed a definition of computational thinking that involves three key dimensions, i.e., computational concepts, computational practices, and computational perspectives. These dimensions may not fully address the actual teaching of CT. However, as Kotsopoulos et al. (2017) noted, they had already captured the what, how, and why of CT.

According to Burke et al. (2019), computational concepts were the fundamental concepts students engaged with as they programmed. While working with Scratch, these include concepts such as sequences, loops, parallelism, automation, conditionals, operators, and data. Next, Burke et al. (2019) defined computational practices as the actual practices students developed as they encountered and engaged with the concepts. These practices took place when students were incremental and iterative, tested and debugged, reused and remixed, as well as abstracted and modularized their Scratch projects. Computational perspectives, as Burke et al. (2019) further explained, were the perspectives students formed about the world around them and about themselves as they comprehended these concepts and engaged in such practices by expressing their thoughts about their works, connecting with others doing similar works through the Scratch community, and questioning themselves as well as other designers about the significance of their works and how these works could be made better.

This study attempted to get a general portrayal of the dimension of the computational concept that the student respondents exhibited in their Scratch projects. Scratch concepts such as sequences, loops, paral-lelism, automation, etc., were analyzed using Dr. Scratch. The results were then described and analyzed to understand better how well the student respondents perceived and applied the concepts within their projects. Whilst there have been a few research about the levels of (pre-service) teachers' computational thinking and metrics, such as Dr. Scratch (Bullee et al., 2020; Kite & Park, 2020; Looi et al., 2020; Troiano et al., 2019), this study aims to give insights into the competence level of computational concepts of pre-service teachers, especially those in the Primary School Teacher Education Program of Sanata Dharma University. The analysis results are hoped to serve as the basis for the integration of CT and their concepts across the curriculum designed for the student teachers. This way, it is hoped that they can effectively infuse their future classes with computational thinking skills, for they have already gained a solid knowledge base of how CT works in their own classes and subjects (Lynch, 2019).

Dr. Scratch

Dr. Scratch is a free/open-source web tool that Hairball powers and that analyzes Scratch projects to automatically assign a CT score in terms of abstraction and problem decomposition, parallelism, logical thinking, synchronization, flow control, user interactivity, and data representation (Moreno-León & Robles, 2015). It can also quickly check if a project has been adequately programmed by allowing users to detect bad coding practices such as repetition of codes, object namings, and incomplete codes. This web tool is therefore useful for students, and also teachers because it allows students to learn from their mistakes and get feedback to improve their code and, by doing so, develop their CT skills (Moreno-León & Robles, 2015; Moreno-León et al., 2015).

Dr. Scratch can analyze projects made by any version of Scratch, i.e., Scratch 1.4, Scratch 2.0, and the newest Scratch 3.0. It provides its users with two ways of analyzing their Scratch projects. The first one is by using the url (uniform resource locator) of the project uploaded to a website, and the other by directly uploading the project to Dr. Scratch website. Either way, Dr. Scratch will provide its users with the project analysis result comprising the score of the project, its level, the bad habits identified in the project, and the level of each CT concept exhibited in the project. It also provides users with a link to download the project certificate, which may be useful for teachers who want to use the Scratch project as one of the ways to acknowledge their students' achievements. The sample result of analysis provided by Dr. Scratch is illustrated in Figure 1.

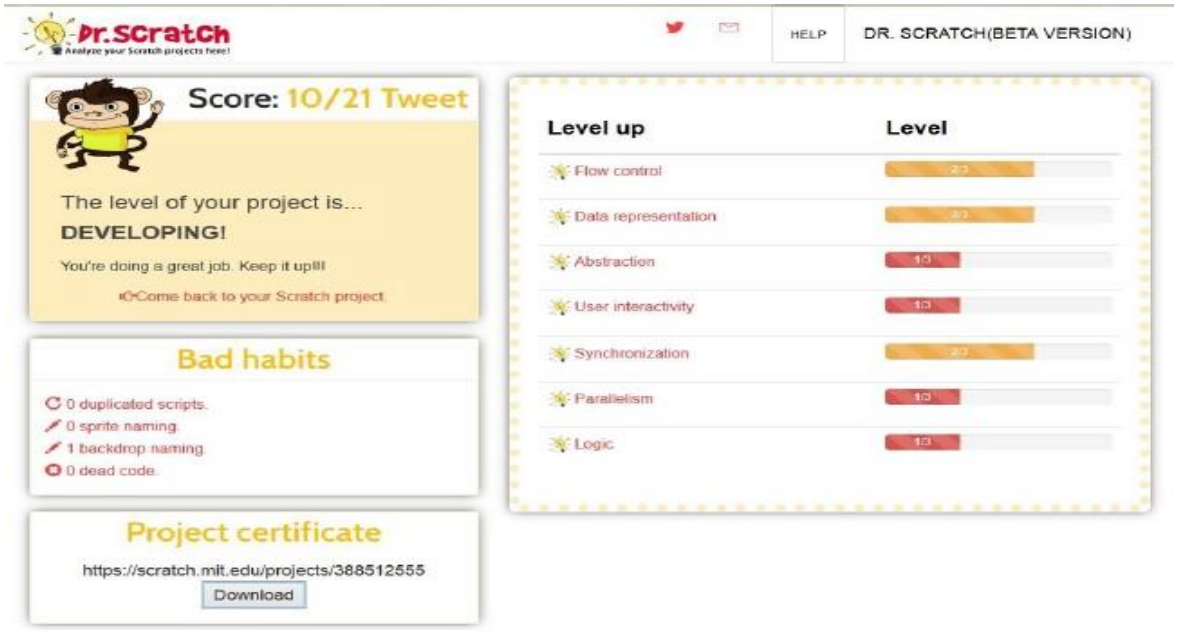

Figure 1. The Sample Analysis Result of a Scratch Project Provided by Dr. Scratch

As shown in Figure 1, Dr. Scratch provides analysis on seven computational concepts commonly exhibited in Scratch projects. Those concepts are flow control, data representation, abstraction, user interactivity, synchronization, parallelism, and logic. Moreno-León et al. (2015) and Rose et al. (2018) detailed the competence levels of each of the concepts as well as the point earned by projects exhibiting them in the Table 1.

Table 1. Competence Levels of CT Concepts in Dr. Scratch

Concept	Basic (1 point)	Develop-ing (2 point)	Master (3 point)
Flow control	Sequence of blocks	Repeat, forever	Repeat until
Data representation	Modifiers of sprites properties	Variables	List
Abstraction	More than one script and more than one sprite	Use of custom blocks	Use of 'clones' (instances of sprites)
User interactivity	Green flag	Keyboard mouse, ask and wait	Webcam, input sound
Synchronization	Wait	Message broadcast, stop all, stop program	Wait until, when backdrop changes to, broadcast and wait
Parallelism	Two scripts on green flag	Two scripts on key pressed or sprite clicked	Two scripts on receive message, video/ audio input, backdrop change
Logic	If	If else	Logic operations

Further, based on the competence levels described in Table 1, Scratch projects are categorized as basic if their score is equal to 7 or less. They are categorized as developing if their score falls between the range of 8 and 14. Scratch projects scored 15 or more are, hence, categorized

as master. This categorization can be used as an indication of how students demonstrate understanding of the concepts as well as the CT skills by applying the concepts into their programs (Kwon et al., 2018).

Apart from its advantageous features, however, Dr. Scratch has few drawbacks. As Šerbec et al. (2018) noted, it could not measure the functionality or creativity of the evaluated Scratch projects. In the first place, it does not check to see if the projects have been modified, remixed, or copied (O'Neill, 2018). Further, Román-González et al. (2017) suggested that Dr. Scratch was only meant as a tool for the formative assessment of Scratch projects. Consequently, it may not be a good choice for teachers wishing to evaluate their student accomplishment in CT skills (O'Neill, 2018).

METHOD

The main purpose of this descriptive research was to describe the competence level of the primary school pre-service teachers' computational concepts as depicted in their Scratch projects using Dr. Scratch. It tried to provide a quick snapshot of how certain computational concepts (Table 1) were exhibited in Scratch projects that the student respondents designed. Further, in line with any descriptive research purpose, it just attempted to describe the prevalence (commonness) of the competence level of computational concepts in the respondents' projects (Adams & Lawrence, 2019).

The respondents were 87 sophomore students of the Primary School Teacher Education Program of Sanata Dharma University chosen using the convenience sampling method. They belonged to four different classes and were among the 235 active sophomore students of the study program. They were the most convenient samples for this study because they belonged to the classes assigned to the researcher and, therefore, their schedules complied with those of the researcher (Battaglia, 2011; Stockemer, 2018).

This research, as mentioned previously, gathered its data from the Scratch projects designed by the student respondents. The projects were assigned as their final projects after the respondents learned how to use Scratch for six weeks as part of their MPBICT (Media Pembelajaran Berbasis ICT) course. The projects were uploaded to the Scratch community (scratch.mit.edu) and then analyzed using Dr. Scratch using the corresponding links shared with the researcher.

The scores of the analysis results provided by Dr. Scratch were calculated to find its mean. This means would show the general level of the respondents' computational concepts. The analysis results would also be categorized based on the seven computational concepts in Table 1 to calculate each of their means. These means would provide a general picture of how the concepts were exhibited in the respondents' projects. As the analysis results also showed the bad programming habits of the projects, the habits were also categorized and counted based on their types to calculate the percentage of each bad coding habit shown in the projects. These percentages would show the type of bad coding habit the respondents commonly made while programming their Scratch projects.

RESULTS AND DISCUSSION

The analysis results of the respondents' Scratch projects showed that from the total of 87 projects, 66 projects were categorized as developing, 20 projects were identified as master, and only one project was labeled as basic. The results indicated that more than three quarters (75.86%) of the projects were scored between 8 and 14, while 22.99% of the projects were scored 15 or more. The only one project (1.15%) categorized as basic received a score of 7 in the analysis performed by Dr. Scratch. These results were in line with what Kite and Park (2020) as well as Looi et al. (2020) suggested that, in general, pre-service teachers did not yet have an adequate level of CT concepts understanding and only a small number of them had an accurate conceptualization of CT. The analysis results also showed the competence level scores of each CT concept identified in the student respondents' Scratch projects. The CT concepts (Table 1) from each submitted project were then organized to calculate their means. The means for each CT concept exhibited in the respondents' projects are presented in Table 2.

Table 2. The Means of CT Concepts Identified in Respondents' Projects

Concept	Means
Flow control	2.56
Data representation	1.95
Abstraction	1.39
User interactivity	1.82
Synchronization	1.94
Parallelism	1.40
Logic	1.47

Table 2 shows that among the seven CT concepts used as indicators of the representation of CT skills in Scratch projects, flow control scores the highest (2.56) while the concept scoring the lowest is an abstraction (1.39). This indicated that flow control was the most common CT concept identified in the respondents' Scratch projects. Likewise, the concept of abstraction became the least exhibited concept in the projects submitted by the respondents (Troiano et al., 2019).

A detailed analysis of the respondent projects revealed that 51 respondents scored 3 in the concept of flow control. The other 34 respondents scored two while the rest two respondents only scored 1 in the same concept. These numbers inferred that most of the respondents (58.62%) had shown more advanced use of the flow control concept in the projects by using the conditional loop command "repeat until." It also indicated that 39.08% of the respondents showed moderate use of the concept through the use of more common loop commands such as "repeat" and "forever" in their projects. Accordingly, it also revealed that somewhat 2.30% of the respondents only showed basic use of the concept as they only created sequences of blocks to execute their programs.

Table 2 also shows that abstraction scores the lowest among the seven CT concepts (1.39). This was evident through the analysis of this concept showing that 67 respondents only scored 1 in the concept. Three respondents scored two while the rest 16 scored 3 in the same concept. This analysis suggested that the majority (77.01%) of the respondents could only repeat using a sprite or an executable script in their projects. Further, 3.45% of the respondents had been able to make their scripts easier to manage and understand by including custom blocks made using "My Blocks." The remaining 18.39%, however, had managed to include clones in their projects. These respondents scored 3 in the concept and, as also suggested by Troiano et al. (2019), had been able to program a sprite to have infinite clones that run at a particular time when the program was executed and immediately executed deleted when they were no longer needed.

Even further analysis of the respondents' projects revealed that those categorized as the master had a wider range of competence levels than those categorized as developing. Though both groups scored the highest on the flow control concept, the concepts on which they scored the lowest were significantly different. While the lowest scored concept for developing projects was abstraction (1.00), this concept got a significantly higher score in master projects (2.70). Table 3 shows the comparison of the means of each CT concept exhibited by each category of projects.

Table 3. The Means of CT Concepts' Identified in Developing and Master Projects

Concepts	Developing	Master
Flow control	2.53	2.75
Data representation	1.95	2.00
Abstraction	1.00	2.70
User interactivity	1.79	1.90
Synchronization	1.79	2.55
Parallelism	1.03	2.60
Logic	1.15	2.60

Table 3 shows that developing and master projects do not show significant differences in the use of flow control, data representation, and user interactivity concepts. Both kinds of projects had shown eminent uses of the loop (e.g., "repeat," "forever," and "repeat until") and conditional blocks,

variables, varied ways of initiating and running the programs using "when clicked," "ask and wait," keyboard keys, mouse, and input sounds as well as and varieties of external image and sound files.

The projects, however, showed some discrepancies in the ways they exhibited the concepts of abstraction, synchronization, parallelism, and logic. Master projects had demonstrated more advanced uses of cloning commands in the control block, broadcast commands in the event block, and logic operations from the operator block. These projects had also included several commands that could initiate at once when a particular event, such as changing the background or receiving messages, took place. Though Dr. Scratch had also identified uses of these commands and blocks in developing projects, they were not as advanced as those found in the master projects. The discrepancies among the competence levels of the CT concepts between the two kinds of projects are illustrated in Figure 2.

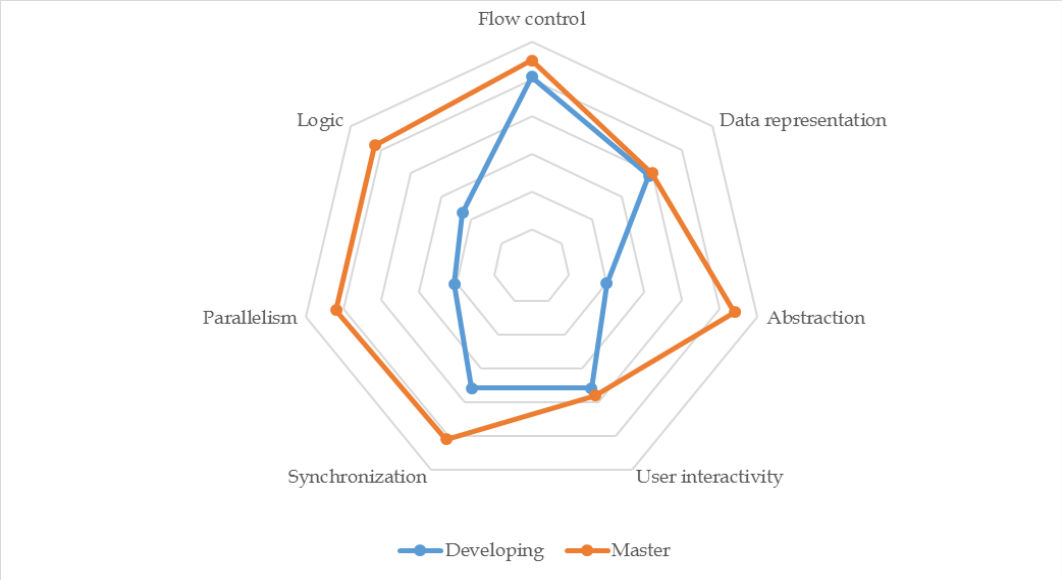

Figure 2. The Discrepancies among the Competence Levels of the CT Concepts in Developing and Master Projects

Analysis from Dr. Scratch showed that both categories of projects had relatively varied scores on their CT concepts. However, both had shown immediate understanding of the concepts and demonstrated essential CT skills by using the concepts in their scripts (Kwon et al., 2018). In the context of the Scratch programming environment, both categories of projects had been able to be used as an indicator of how the student respondents made use of the concepts of sequences, data, loops, automation, parallelism, operators, and conditions (Brennan & Resnick, 2012; Bullee et al., 2020; Fagerlund et al., 2021).

As previously mentioned, Dr. Scratch also provides analysis on bad programming habits commonly identified in Scratch projects. Thereby, this study also organized these habits to find out the number of their occurrences. This would show what kind of bad programming habits were mostly made by the respondents in their projects. Each bad habit and the number of projects exhibiting it are presented in the Table 4.

Table 4. The Number of Projects with Bad Programming Habits

Bad Habits	Number of Projects	Percentage
Duplicated scripts	46	52.87
Sprite namings	18	20.69
Backdrop namings	35	40.23
Dead code	11	12.64

In line with what Moreno-León and Robles (2015) and Moreno-León et al. (2015) had suggested, Table 4 also showed that the most common bad habit that the student respondents made in their Scratch projects were duplicated scripts. As depicted in the table, Dr. Scratch had identified duplicated scripts in more than half of the projects. These duplicated scripts were mostly in the form of two or more programs having the same blocks that only varied in their parameters or values.

Incorrect namings, i.e., sprite and backdrop namings, were the nest common bad habits made by the respondents. Bad sprite namings were identified in 18 projects, while bad backdrop namings were found in 35 projects. In projects with more than one sprite and one backdrop, bad namings were typically identified when the sprites or backdrops were not appropriately labeled. In these cases, they were usually just left with the default names Sprite 1, Sprite 2, Backdrop 1, Backdrop 2, etc., instead of being labeled with names that showed their specific characteristics in the projects, e.g., flower, apple, Sandy, etc. Proper namings are supposed to make the programs be read faster and, therefore, will be really handy if the programs need debugging (Moreno-León & Robles, 2015; Moreno-León et al., 2015).

Finally, dead code was the least common bad programming habit identified in the respondents' projects as it was only found in 12.64% of the total submitted projects. This meant that the majority of the projects (87.36%) did not have codes that were not executed in their programs. Most of the codes used in these projects had presumably run properly and so did not cause them to run into problems when executed.

In terms of their number of occurrences, it was foreseeable that duplicated scripts were also the most common bad programming habits made by the respondents in their projects. It made up 39.31% of the total bad programming habits identified. Bad sprite namings made up 29.48%, and backdrop namings made up 23.70% of the bad habits. Dead codes were still the least common bad programming habits made as they only made up 7.51% of the total bad habits made by the respondents.

When both categories of projects were compared, it was found that developing projects had more duplicated scripts, bad background namings, and dead codes than master projects. However, Dr. Scratch had identified more bad sprite namings in master projects than in developing ones. The number of bad sprite namings identified in master projects was almost threefold those identified in developing projects. The comparison of each bad programming habit between the two categories of projects is presented in Table 5.

Table 5. The Comparison of Bad Programming Habits between Developing and Master Projects

Bad Habits	Developing	Master
Duplicated scripts	62	6
Sprite namings	13	38
Backdrop namings	32	9
Dead code	9	4

Table 5 indicates that, in general, projects categorized as developing had more bad programming habits than projects categorized as master. However, compared to developing projects, master projects had apparently more bad sprite namings than developing ones. Presumably, this was related to the high scores of their CT concepts of abstraction, synchronization, and parallelism. As respondents' master projects tended to have multiple scripts executed simultaneously, it was doubtless that they had multiple sprites too. These sprites might be executed concurrently using a single command or consecutively using commands within "events" and "control" blocks. As there were more scripts, there were reasonably more sprites used in them.

As detailed as it was, however, the analysis result provided by Dr. Scratch could only infer the what of CT skills in Scratch (Kotsopoulos et al., 2017). Dr. Scratch can effectively classify certain projects as basic, developing, or master. Nevertheless, it could not clarify how and why the student respondents exemplified the CT concepts in their projects. It could not identify whether the projects were ingenious or whether they were only copied and remixed from other projects (O'Neill, 2018). For this reason, it was apparent that the analysis result from Dr. Scratch could not assess the creative

elements of the evaluated projects (Šerbec et al., 2018). As such, instead of being used as an indicator of the respondents' real performance in their CT skills, the analysis results provided by Dr. Scratch would be an ideal tool to assess progress in the development of the student teachers' computational concepts in Scratch.

CONCLUSIONS

Based on the analysis results provided by Dr. Scratch, the CT skills of the primary school pre-service teachers of Sanata Dharma University, as depicted in their Scratch projects was still at the level of development. This was supported by the analysis results showing that 75.86% of the Scratch projects submitted by the student respondents were classified into this category. The projects categorized as master made up 22.99% of all the submitted projects, while those categorized as basic was only 1.15% of all the projects. Based on the findings of this study, a few propositions can be put forward as recommendations. First, the primary school pre-service teachers of Sanata Dharma University could be provided with more opportunities to enhance their CT skills by attending and engaging in courses or activities that can help them develop and improve their skills. Next, as this research mainly serves as preliminary research about the dimension of computational concepts in CT skills, further research is necessary to diagnose and qualify the actual picture of the pre-service students' CT skills. In the context of the Scratch programming environment, future research should focus on the three dimensions of CT skill by providing a more detailed account of what concepts the students have been able to include in their projects. Moreover, they will also need to provide a clearer understanding of the computational practices and computational perspectives of the students as they include the concepts in their programs..

REFERENCES

Adams, K. A., & Lawrence, E. K. (2019). *Research methods, statistics, and applications* (2nd ed.). SAGE Publications, Inc.

Angevine, C. (2018). *Advancing computational thinking across K-12 education*. Gettingsmart.Com. https://www.gettingsmart.com/2018/02/advancing-computational-thinking-across-k-12-education/

Barr, D., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *Learning & Leading with Technology*, *38*, 20–23. https://files.eric.ed.gov/fulltext/EJ918910.pdf

Battaglia, M. P. (2011). Nonprobability sampling. In *Encyclopedia of Survey Reasearch Methods*. Sage Publications.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Annual American Educational Research Association Meeting*, 1–25. http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

Bullee, A., Norbert, A., Naseeven, P., Pultoo, A., Oojorah, A., Roocha, V., Sheoraj, K., Rajcoomar, H., Panchoo, S., & Ujoodha, M. (2020). Codecraft competition: Learning to code through contests using scratch. *Journal of Science and Technology*, *5*(4), 40–53. https://doi.org/10.46243/jst.2020.v5.i4.pp40-53

Burke, Q., Bailey, C. S., & Ruiz, P. (2019). *CIRCL primer: Assessing computational thinking*. CIRCL Primer Series. http://circlcenter.org/assessing-computational-thinking

Elskamp, F. (2018). *CoDuo: A game for teaching computational thinking in primary education*. http://essay.utwente.nl/75552/

Fagerlund, J., Häkkinen, P., Vesisenaho, M., & Viiri, J. (2021). Computational thinking in programming with Scratch in primary schools: A systematic review. *Computer Applications in Engineering Education*, *29*(1), 12–28. https://doi.org/10.1002/cae.22255

Fingal, J. (2018). *Teaching computational thinking more important than defining it*. Iste.Org. https://www.iste.org/explore/Computational-Thinking/Teaching-computational-thinking-more-important-than-defining-it

Hunsaker, E. (2020). Computational Thinking. In A. Ottenbreit-Leftwich & R. Kimmons (Eds.), *The K-12 Educational Technology Handbook*. EdTech Books. https://edtechbooks.org/k12handbook/computational_thinking/simple

ISTE, & CSTA. (2011). *Operational definition of computational thinking for K–12 education*. National Science Foundation.

Jacob, S., & Warschauer, M. (2018). Computational thinking and literacy. *Journal of Computer Science Integration*, *1*(1), 1–19. https://doi.org/10.26716/jcsi.2018.01.1.1

Kite, V., & Park, S. (2020, March 26). Secondary science teachers conceptualizations of computational thinking and perceived barriers to CT/content integration. *The 2020 Annual Meeting of the National Association for Research in Science Teaching (NARST)*. https://www.researchgate.net/publication/340175597

Kotsopoulos, D., Floyd, L., Khan, S., Namukasa, I. K., Somanath, S., Weber, J., & Yiu, C. (2017). A pedagogical framework for computational thinking. *Digital Experiences in Mathematics Education*, *3*(2), 154–171. https://doi.org/10.1007/s40751-017-0031-2

Kwon, K., Lee, S. J., & Chung, J. (2018). Computational concepts reflected on Scratch programs. *International Journal of Computer Science Education in Schools*, *2*(3). https://doi.org/10.21585/ijcses.v2i3.33

Looi, C.-K., Chan, S. W., Huang, W., Seow, P., & Wu, L. (2020). Preservice teachers' views of computational thinking: STEM teachers vs non-STEM teachers. In S. C. Kong, H. U. Hoppe, T. C. Hsu, R. H. Huang, B. C. Kuo, K. Y. Li, C. K. Looi, M. Milrad, J. L. Shih, K. F. Sin, K. S. Song, M. Specht, F. Sullivan, & J. Vahrenhold (Eds.), *The Fourth International Conference on Computational Thinking Education 2020* (pp. 73–76). The Education University of Hong Kong. https://www.researchgate.net/publication/343737473_Preservice_Teachers'_Views_of_Computational_Thinking_STEM_Teachers_vs_non-STEM_Teachers

Lynch, M. (2019). *Why we must teach our teachers computational thinking*. Thetechedvocate.Org. https://www.thetechedvocate.org/why-we-must-teach-our-teachers-computational-thinking/

Martins-Pacheco., L., von Wangenheim., C., & Alves., N. (2019). Assessment of computational thinking in K-12 context: Educational practices, limits and possibilities - A systematic mapping study. *Proceedings of the 11th International Conference on Computer Supported Education - Volume 1: CSEDU*, 292–303. https://doi.org/10.5220/0007738102920303

McVeigh-Murphy, A. (2019). *What is computational thinking? and why is it important for students?* Equip.Learning.Com. https://equip.learning.com/computational-thinking

Moreno-León, J, & Robles, G. (2015). *Analyze your Scratch projects with Dr . Scratch and assess your Computational Thinking skills*.

Moreno-León, Jesús, Robles, G., & Román-González, M. (2015). Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *Scratch Conference*, 12–15. https://jemole.me/replication/2015scratch/InferCT.pdf

O'Neill, J. (2018). *SPAE: A Scratch Project Analysis tool for Educators* [Appalachian State University]. http://libres.uncg.edu/ir/asu/f/O%27Neill_Joseph_2018_Thesis.pdf

Papert, S. A. (1993). *Mindstorms: Children, computers, and powerful ideas* (2nd ed.). Basic Books.

Rich, K. M., Yadav, A., & Larimore, R. A. (2020). Teacher implementation profiles for integrating computational thinking into elementary mathematics and science instruction. *Education and Information Technologies*, *25*(4), 3161–3188. https://doi.org/10.1007/s10639-020-10115-5

Román-González, M., Pérez-González, J.-C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior*, *72*, 678–691. https://doi.org/https://doi.org/10.1016/j.chb.2016.08.047

Rose, S., Habgood, J., & Jay, T. (2018). Pirate plunder: Game-based computational thinking using scratch blocks. In *Proceedings of the 12th European Conference on Games Based Learning* (pp. 556–564). Academic Conferences and Publishing International Limited. http://shura.shu.ac.uk/21715/

Šerbec, I. N., Cerar, Š., & Žerovnik, A. (2018). Developing computational thinking through games in Scratch. *Education and Research in the Information Society*. http://hdl.handle.net/10525/2943

Stockemer, D. (2018). *Quantitative methods for the social sciences: A practical introduction with examples in SPSS and stata*. Springer International Publishing. https://doi.org/10.1007/978-3-319-99118-4

Troiano, G. M., Snodgrass, S., Arg\imak, E., Robles, G., Smith, G., Cassidy, M., Tucker-Raymond, E., Puttick, G., & Harteveld, C. (2019). Is my game ok Dr. Scratch? Exploring programming and computational thinking development via metrics in student-designed serious games for STEM. *Proceedings of the 18th ACM International Conference on Interaction Design and Children*, 208–219. https://doi.org/10.1145/3311927.3323152

Vaidyanathan, S. (2016). *What's the difference between coding and computational thinking?* Edsurge.Com. https://www.edsurge.com/news/2016-08-06-what-s-the-difference-between-coding-and-computational-thinking

Vatansever, Ö., & Baltaci Goktalay, S. (2018). How does teaching programming through Scratch affect problem-solving skills of 5th and 6th grade middle school students? *International Journal of Management and Social Sciences*, *9*(33), 1778–1801. https://www.researchgate.net/publication/328601938

Voinohovska, V., Tsankov, S., & Goranova, E. (2019). Development of the students' computational thinking skills with project-based learning in Scratch programming environment. *12th International Conference on Education and New Learning Technologies*, 5254–5261. https://doi.org/10.21125/edulearn.2020.0133

Wang, P. S. (2017). *From computing to computational thinking* (1st ed.). CRC Press. https://doi.org/10.1201/9781315115320

Weese, J. L., & Feldhausen, R. (2017). STEM outreach: Assessing computational thinking and problem solving. *2017 ASEE Annual Conference & Exposition*. https://doi.org/10.18260/1-2--28845

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