The Role of Scratch Visual Programming in the Development of Computational Thinking of Non-IS Majors

Completed Research Paper

Slade Scullard **Pitso Tsibolane**

Malcolm Garbutt

Abstract

The study explored the role of Scratch in developing the computational thinking (CT) abilities of Non-IS majors. Literature shows that abstraction, parallelism, logical thinking, data representation, flow control, pattern generalization and systematic processing of information produce computational thinking. Using a survey (n = 92) analyzed through PLS-SEM, the study explored and validated computational thinking definitions and constructs based on the other constructs. A final conceptual model shows the relationships between the constructs. The results of the survey indicated that Scratch played a significant role in abstraction for developing computational thinking. Further analysis concluded that Scratch also played a role in developing logical thinking by acting through abstraction and the other CT constructs. Nevertheless, these were not observed to influence computation thinking significantly. Further research is required to link logical thinking to computational thinking and to determine if flow control has a mediating or moderating impact on computational thinking.

Keywords: Computational Thinking, Programming, Scratch, Non-IS Students, Education

Introduction

Computational Thinking (CT) is fundamental to computer science as it has paved the theoretical foundation for applications and systems that solve complex human problems. Educators and experts in the technology field have highlighted that computational thinking is an important skill to learn (Voogt, Fisser, Good, Mishra, & Yadav, 2015) especially in the 21st century where computers are used in almost every environment (Kalelioğlu & Gulbahar, 2014; Djambong & Freiman, 2016; Korkmaz, Çakir & Ozden, 2017).

The importance of computational thinking has led to multiple studies for teaching computational thinking skills and how to promote and assess the development of computational thinking. A conventional method of teaching computational thinking skills is through programming languages

especially visual programming languages such as Scratch (Weintrop & Wilensky, 2015). Although Scratch visual programming is taught at various well-known universities such as MIT and Harvard as well as other universities throughout the world as an introductory programming language that can enhance student's computational thinking (Brennan & Resnick, 2012). This raises the questions of students' perceptions of the benefits of learning Scratch and how the Scratch visual programming language contributes to the development of computational thinking. Hence, the purpose of this research was to explore perceptions of students, particularly non-IS majors, towards Scratch as an introductory programming language and an enabler of computational thinking. The research also sought to uncover the same students' overall perceptions about the process of learning Scratch visual programming.

The paper proceeds as follows. The next section provides a definition of computational thinking and an overview of teaching and assessing computational thinking through learning Scratch visual programming. In the third section, the research methods for the findings presented in section four are described. Section five concludes the study and provides limitations and suggested future research.

Background to Computational Thinking and Scratch

High-levels of computerization imposes computational abilities of employees and increases the demand for computational thinking (Lockwood and Mooney, 2017). Furthermore, the adoption of digital computing technologies transforms job requirements making the development of computational thinking skills imperative (Kale et al., 2018).

Although controversial (Lawanto, Close, Ames, & Brasiel, 2017), literature defines computational thinking as an approach to solving problems, designing systems and understanding human behavior. In contrast to other forms of thinking, computational thinking draws on concepts and thought processes fundamental to computing (Wing, 2006). Voogt et al. (2015) elaborated on this definition, describing computational thinking as a universal attitude and skill set that includes decomposition, abstraction, algorithmic thinking and pattern matching and many more. Consequently, computational thinking is considered as a thought process critical for solving problems in a technology-driven society (Kale et al., 2018). It is also considered a skill incorporating understanding problems and the ability to logically analyze data and systematically identify and implement a solution (CSTA & ISTE, 2011).

With combinations of problem-solving skills and systematic-thinking, computational thinking could improve students' abilities in areas beyond computing (Lockwood & Mooney, 2017). For example, across industries, students who grasp and effectively use computational thinking as a skill habitually produce more effective solutions than those who are not able to think computationally (Korkmaz et al., 2017). Studies have shown that incorporating computational thinking into teaching methods improves analytical ability and can be used as an early predictor of academic achievement in students (Lockwood & Mooney, 2017). Analytical thinking includes mathematical problem solving, an engineering approach to design and evaluation, and a scientific approach to understanding computational intelligence, the mind and human behavior (Wing, 2008).

Even with this comprehensive definition, which has been both extended and criticized as shown in Table 1, the literature on computational thinking is immature and fails to adequately explain computational thinking or how it can be effectively taught and assessed (Kalelioğlu, Gülbahar, & Kukul, 2016).

Table 1. A Sample Table

Author(s)	Computational Thinking Definition
Wing (2006)	Computational thinking involves solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science.
Wing (2008)	Computational thinking is a kind of analytical thinking. It shares with mathematical thinking in the general ways in which we might approach solving a problem.
Wing (2011)	Computational Thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent.

Sengupta, Kinnebrew, Basu, Biswas and Clark (2013)	Computational thinking encompasses being able to distinguish several levels of abstraction and apply mathematical reasoning and design-based thinking.
Korkmaz et al. (2017)	Computational thinking is an essential skill that comes to the meaning of problem-solving for the human beings and points out that it is necessary to understand what the problem is before thinking of the solutions while solving a problem according to a certain point of view.
	Computational thinking is having the knowledge, skill and attitudes necessary to be able to use the computers in the solution of the life problems for production purposes.
Kale et al. (2018)	Computational thinking is one of the skills critical for successfully solving problems posed in a technology-driven and complex society.

Several studies and models have been proposed to teach computational. Brennan and Resnick (2012) provide a computational thinking framework comprising computational concepts, computational practices and computational perspectives. Lye and Koh (2014) expanded the Brennan and Resnick (2012) framework to include loops and recursion in the computational concepts. According to the Computer Science Teachers Association and the International Society for Technology in Education (CSTA & ISTE, 2011), computational thinking is a problem-solving process that includes:

- Understanding and defining problems in a manner which enables humans to use computers to solve them
- Logically organising and analysing data
- Representing data through abstraction
- Automating solutions through algorithmic thinking
- Identifying, analysing, and implementing solutions that aim to achieve the most efficient and effective combination of steps and resources.

Nevertheless, these definitions do not explain how to think computationally nor how to teach computational thinking.

Computational thinking in education

Teaching and assessing computational thinking skills in education remains unclear (Korkmaz et al., 2017). Computer programming has been proposed for teaching computational thinking as application development is more than just a fundamental skill and is a key tool for catalysing cognitive tasks involved in computational thinking. Additionally, developing applications demonstrates computational abilities (Grover & Pea, 2013). The ability to transfer ideas into representations that leverage computational power is central to computational thinking. Thus, computational thinking can be analysed by understanding the mental task of programming and applying the programming language constructs and computer algorithms (Weintrop & Wilensky, 2013).

Computational thinking is an underlying cognitive process that allows code literacy which entrenches the view that programming is fundamental to enable computational thinking (Lye & Koh, 2014). Several studies show that programming can improve students' higher-order thinking skills and help them to develop algorithmic approaches to solving problems aligns with the essence of computational thinking (Israel, Pearson, Tapia, Wherfel & Reese, 2015). According to Israel et al. (2015) both experience and computational thinking is increased in science, engineering and mathematical areas of studies when programming is introduced and integrated into learning environments. This resonates with the claims that programming languages like Logo, used in the 1980s and 1990s, could develop thinking skills (Lye & Koh, 2014). However, there is a lack of conclusive evidence of Logo improving these skills (Lye & Koh, 2014). Grover and Pea (2013) discuss several alternative programming tools such as Alice, Game Maker, Kodu, Greenfoot and Scratch that encourage computational thinking in novice students. Visual programming languages such as Scratch, Alice and Kodu have been designed to allow

students to code without the need to learn syntax making it ideal for students new to programming (Moreno-León & Robles, 2016).

Relationships between the systematic processing of information (SYS), pattern generalization (PATG) and parallelism (PARA) were also explored. Each affords explanations of the systematic processing of information that can induce a level of pattern generalization which in turn requires the use of parallelism. The ability to understand information with deep thinking and reasoning allows the learner to link several separate pieces of information thereby generalizing data patterns (Moreno-León & Robles, 2015; Lawanto et al., 2017). These constructs could highlight sequences and constraints which use parallel thinking processes to assess similar problems and underlying patterns to form a solution for other problems (Grover & Pea, Brennan & Resnick, 2012; 2013; Djambong & Freiman, 2016; Lawanto et al., 2017; Kale et al., 2018).

Nevertheless, what constitutes computational thinking must first be defined before exploring how to teach computational thinking.

Towards a definition of computational thinking

Researchers suggest a range of distinct conceptual constructs that make up computational thinking. Brennan and Resnick, (2012) highlighted seven concepts which can be identified in both programming and non-programming contexts; sequences, loops, parallelism, events, conditionals, operators, and data. Grover and Pea (2013) include conceptual constructs of computational thinking such as abstraction and pattern generalisation; systematic processing of information; structured problem decomposition; iterative, recursive, and parallel thinking; conditional logic form the basis of learning and assessing computational thinking. Moreno-León and Robles, (2015) define key conceptual constructs of computational thinking as abstraction, logical thinking, synchronisation, parallelism, flow control, user interactivity and data representation. Voogt et al. (2015) described core computational thinking concepts that could be used in education including data collection, data analysis, data representation, problem decomposition, abstraction, algorithm and procedures, automation, parallelisation and simulation. Lawanto et al. (2017) based their research on abstraction, parallelism, logical thinking, synchronization, flow control, user interactivity and data representation.

Drawing from these elements, we define computational thinking as a thought process that is critical for solving problems in a technology-driven society (Kale, et al., 2018) which seeks to understand problems and to analyse data logically, and to identify and implement solutions in a systematic way. Thus, computational thinking encompasses the core skills of abstraction, parallelism, logical thinking, data representation, flow control, pattern generalization, and systematic processing of information. Based on these understandings of computational thinking, the conceptual constructs that constitute computational thinking are summarised in Table 2 as a conceptual framework.

Table 2 - Conceptual Thinking Conceptual Framework

Conceptual element	Definition	Author
Abstraction	Abstraction is the ability to break down information and work tasks into smaller tasks so that it is easier to understand and easier to work with. It also explains the skill to find the most important information that is necessary to solve a problem.	(Brennan & Resnick, 2012; Lawanto et al., 2017).
	Abstraction is the ability to reduce information and detail to focus on concepts relevant to understanding and solving problems.	(Grover & Pea, 2013; Rose, Habgood & Jay, 2017)
	Abstraction is the ability to capture different angles of approach to a problem and its solution.	(Djambong & Freiman, 2016).
Parallelism	Parallelism is the ability to think along channels where the focus of the problem is split into more than one direction and involves many reiterations.	(Brennan & Resnick, 2012; Lawanto et al., 2017).

	Parallelism is the ability to have more than one thing happening at once in a thinking or actionable process.	(Rose et al., 2017).
Logical thinking	Logical thinking is the ability to use conditional thinking where the solution to the problem needs to make decisions to be successful. It involves the 'if-then-else" construct whereby the thinking should allow this to happen if that is the case or else then that should happen.	(Grover & Pea, 2013; Lawanto et al., 2017).
Data representation	Data representation is the ability to show the correct order of code so that it makes sense and allows the solution to run correctly.	(Voogt et al., 2015; Lawanto et al., 2017).
	Data representation is the ability to use and analyze data to help solve a problem.	(Rose et al., 2017).
Flow control	Flow control is the ability to plan a set of actions for events and develop instructions that flow in a logical manner to make up the solution to the problem.	(Grover & Pea, 2013; Lawanto et al., 2017)
Pattern generalization	Pattern generalization is the ability to identify patterns in data and represent them in a manner which highlights different sequences and constraints that clarify the solution to the problem.	(Grover & Pea, 2013; Kale et al., 2018).
	Pattern generalization is the ability to link a specific problem to other problems of the same type that has already been solved realize that the solution to a given problem, can be the basis of the resolution of a wide range of similar problems.	(Djambong & Freiman, 2016)
Systematic processing of information	Systematic processing of information is the ability to understand available information with deep thinking and reasoning while linking one piece of information to another that clarifies the problem.	(Moreno-León & Robles, 2015; Lawanto et al., 2017)

Scratch as an enabler of computational thinking

Scratch is a block-based visual programming language environment that was inspired by Lego blocks (Resnick et al., 2009). Students assemble code via dragging and dropping instructions to form blocks of code (Weintrop & Wilensky, 2015). The error-free code and user-friendly interfaces of Scratch reduces the complexity of programming for novice programmers (Lai & Yang, 2011). Students use the Scratch environment to program interactive stories, games and animations (Kalelioğlu & Gülbahar, 2014) by mapping together collections of visual "programming blocks" to create programs.

Scratch has a "low floor" - it is easy to start programming with - and a "high ceiling" – it allows users to create complex projects (Resnick et al., 2009; Su, Huang, Yang, Ding, & Hsieh, 2015, Lawanto et al., 2017). Scratch also has "wide-walls" capable of supporting a variety of projects that engage a broad range of people with different interests (Resnick et al., 2009). The drag-and-drop mechanism, together with the natural language labels on the blocks, the shapes and colours of the blocks and the use of a block library makes it easier for students to develop computational thinking skills than with traditional programming languages such as Java (Weintrop & Wilensky, 2015; Lawanto et al., 2017).

The use of Scratch is deemed to benefit students' learning of mathematical and computational concepts while improving their reasoning skills and the ability to work in collaboration with other students (Lawanto et al., 2017). These skills are believed to carry over to non-programming domains while providing opportunities for reflecting on one's thinking and even thinking about thinking itself (Resnick et al., 2009). Thus, the capability of Scratch to enhance the computational thinking abilities of students supports the use of Scratch as an enabler of computational thinking and an ideal platform for introducing Non-IS majors to application development and consequently as the platform used in this study (Lai & Yang, 2011; Weintrop & Wilensky, 2015; Moreno-León & Robles, 2016).

Based on the proposed computational thinking conceptual model, seven initial hypotheses were provided for the testing of the computational thinking efficacy of the Scratch programming language. The conceptual framework for constructs that comprise computational thinking and study hypotheses is depicted in Figure 1.

H1 - Learning Scratch programming contributes to student's abstraction (ABS) abilities.
H2 - Learning Scratch programming contributes to student's parallelism (PARA) abilities.
H3 - Learning Scratch programming contributes to student's logical thinking (LOGIC) abilities.
H4 - Learning Scratch programming contributes to students' data representation (DATAR) abilities.
H5 - Learning Scratch programming contributes to students' flow control (FLC) abilities.
H6 - Learning Scratch programming contributes to students' pattern generalisation (PATG) abilities.
H7 - Learning Scratch programming contributes to students' systematic processing of information (SYS) abilities.

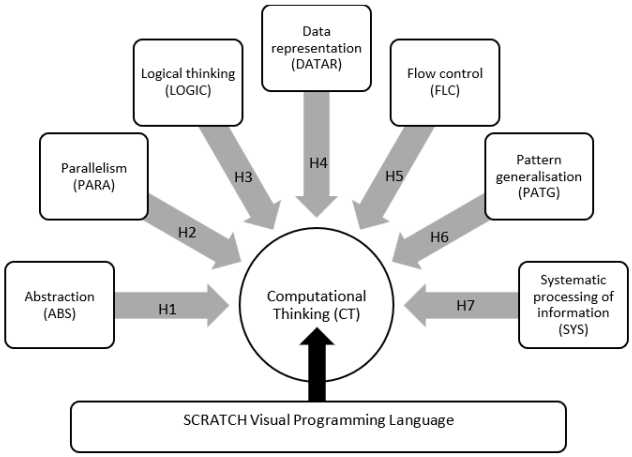

Figure 1 - Computational thinking conceptual construct model

Research Methodology

To test the hypotheses, a positivist cross-sectional quantitative survey was developed based on the conceptual framework. A deductive analysis was undertaken using an online questionnaire with a combination of multiple choice and open-ended questions. The questionnaire was designed in Qualtrics, an online survey tool that was also used to collect data. Qualtrics allows both quantitative and qualitative data gathering. The first of three sections to the questionnaire consisted of five demographic questions. The second section consisted of seven questions in the form of a 5-point Likert Scale that related to their interactions with Scratch as well as student perceptions of Scratch as an introductory programming language. The final section consisted of questions related to seven factors or constructs with each factor containing three questions (in the form of a 5-point Likert Scale) and one open-ended question which make up a total of 28 items.

The survey respondents were acquired through probability sampling (Saunders et al., 2009) from cohorts of Non-IS majors who have completed a Scratch course at the University. Non-IS majors were deemed to be less biased, more honest and critical about their learnings than IS-Majors whose knowledge of other programming languages could impact their learning-perceptions of Scratch. The sample consisted of students that are in their 2nd, 3rd and 4th year of studying Non-IS related degrees.

The data was exported from Qualtrics into IBM SPSS where the data was checked, and appropriate headings were created. The reviewed data from IBM SPSS was exported into an Excel spreadsheet and imported into SmartPLS 3.2.7 which was used for data analysis using Partial Least Squares Structural Equation Modelling (PLS-SEM). PLS-SEM supports the use of small sample sizes (Hair, Ringle & Sarstedt, 2013; Roky & Meriouh, 2015) and the use of reflective models where the normality of the data is not required (Roky & Meriouh, 2015). The study used a reflective model which required an

evaluation of the measurements (outer model) followed by an evaluation of the structural model (inner model) (Hair, Sarstedt, Hopkins & Kuppelwiser, 2014). The structural model linked all independent conceptual elements of abstraction, parallelism, logical thinking, data representation, flow control, pattern generalization and systematic processing of information link to the dependent construct of computational thinking.

Findings

Demographics

The survey was undertaken by 106 students of which 92 questionnaires were deemed complete and acceptable. Of the complete responses, 52% (n=48) were female and 46% (n=42) were male with two responders (2%) not answering the gender question. Most of the respondents were registered for the degrees of Bachelor of Business Science (BBusSci) (40%, n=37) and Bachelor of Commerce (BCom) (35%, n= 32). Accounting (35%) and Finance (16%) majors from the Faculty of Commerce accounted for the majority of subject majors of the respondents. Most respondents were in their second year of study (52%, n=47) and since these students had completed the Scratch course in the year prior to the study they had a more recent experience with Scratch than the other respondents. For the most part (80%, n=74), Scratch was the respondents' first programming experience and thus were not unduly influenced by a-priori computational thinking skills.

Student perceptions of the benefits of learning Scratch

A numeric analysis was undertaken to determine the perceptions of Non-IS Major students of the benefits of learning Scratch. The results are presented in Table 3.

Table 3. Student perceptions of the benefits of learning Scratch.

Construct	Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree
Enjoyable Experience	8	13	11	43	17
Interest in Computers and Programming	8	18	17	36	13
Enriched Academic Life	10	29	29	23	1
Confidence for Computational Tasks	9	22	19	36	6
Understanding Computers	8	24	18	35	7
Remain in Curriculum	6	13	20	37	16
Important for Non-IS Majors	2	13	12	40	34

Approximately 65% (n=60) of students agreed that learning Scratch was a pleasing experience. Frequency results showed a slightly above-average positive attitude towards curiosity in computers and programming affected by Scratch (53%, n=49). The most significant negative finding in respect of Non-IS students' perceptions of Scratch was in the perception of enriching academic life. According to the survey, only 26% (n=24) agreed with this sentiment. One respondent said: "In all honesty, it has not really helped me with other tasks" with another respondent saying: "It has had no effect on anything in my life besides my IS mark".

Limited agreement was also found with regards to confidence in tackling other computational tasks. Only 46% (n=42) agreed to this statement with one respondent saying: "I find the value to be in the practice of simplification and subsequent increased confidence in tackling more demanding and multifaceted projects in other courses". Likewise, 46% (n=42) agreed with the perception that learning Scratch was important in understanding how computers worked. One respondent said: "Learning Scratch enabled me to structure information in a way that I can understand and so that the computer can understand".

A small majority of respondents (58%, n=53) indicated that Scratch should remain part of the Non-IS university curriculum to introduce programming skills. According to one respondent: "I really enjoyed

messing around with Scratch due to the limiting nature of the block-by-block style. It added a weird challenge to some things that showed me alternate ways of doing things, while still being as efficient as possible". This was construed to be supported by the 65% (n=60) of respondents who indicated that learning Scratch was an enjoyable experience and the 80% (n=74) who believe that learning Scratch is important for Non-IS Majors. One respondent said: "It has taught me the language that computers use to complete tasks and how they can have a huge impact in your life regardless the profession that you are going to work in".

Conceptual Model Testing

The conceptual model's reliability and validity were tested for internal consistency and reliability based on the interrelationship of indicator variables observed in the results (Hamid, Sami & Sidek, 2017) and depicted in Table 4. Cronbach alpha and Composite Reliability are the most common measures for internal consistency (Hamid et al., 2017). Whereas the Cronbach Alpha is traditionally used, Composite Reliability provides a better measure of internal consistency (Hair et al., 2014). These measures show the degree to which the construct variables indicate the construct itself (Alshibly, 2014). In exploratory research, the values of these measures are between 0 and 1; where a value closer to 1 indicates higher reliability. A value higher than 0.7 is deemed to be acceptable in PLS-SEM (Hamid et al., 2017). Both the Cronbach Alpha and the Composite Reliability values of each computational thinking construct is higher than 0.7 which indicates a reliable internal consistency.

To reflect Indicator Reliability the proportion of variance in the construct indicator variance that is explained by the construct (Hamid et al., 2017) and indicated by the outer loadings of the conceptual model should be higher than 0.7 (Hamid et al., 2017). The cross-loadings for each construct and its indicators were all above 0.7 indicating adequate Indicator Reliability.

Convergent validity - the extent to which the construct indicators measuring the same construct agree with each other (Alshibly, 2014) - for which the Average Variance Extracted (AVE) was used (Alshibly, 2014; Hamid et al., 2017) - showed the outer model in this study to be adequate with all AVE values above the recommended value of 0.5 (Alshibly, 2014; Roky & Meriouh, 2015; Hamid et al., 2017).

Discriminant Validity - the degree to which the constructs differ from each other and indicated by the low correlation between the construct in question and other constructs (Alshibly, 2014; Hair et al., 2014; Hamid et al., 2017) – showed the cross-loadings for each indicator as well as the Fornell-Larcker criterion - indicating distinctness of a construct (Alshibly, 2014; Hair et al., 2014; Hamid et al., 2017) - of each construct indicator on the same construct as higher than the loadings on other constructs, which indicated that the constructs were distinct (Alshibly, 2014; Hamid et al., 2017).

Table 4 - Construct Reliability and Validity Estimates.

Factor/Reliability	**ABS**	**CT**	**PARA**	**LOGIC**	**DATAR**	**FLC**	**PATG**	**SYS**	**AVE (< 0.5)**
ABS	**0.90**								0.810
CT	0.74	**0.83**							0.686
PARA	0.57	0.58	**0.87**						0.755
LOGIC	0.68	0.65	0.83	**0.86**					0.735
DATAR	0.67	0.57	0.70	0.81	**0.91**				0.834
FLC	0.64	0.60	0.68	0.77	0.82	**0.89**			0.786
PATG	0.65	0.58	0.63	0.77	0.74	0.77	**0.87**		0.755
SYS	0.49	0.49	0.65	0.72	0.68	0.74	0.76	**0.85**	0.727
Note: The bold elements on the diagonal represent the square roots of the average variance extracted, and off-diagonal elements are the correlation estimates.									
Composite Reliability	0.927	0.867	0.902	0.893	0.938	0.917	0.902	0.888	

Cronbach's Alpha	0.882	0.771	0.837	0.820	0.901	0.864	0.840	0.814	

Structural Model Results

Based on the reliability of the data, the study evaluated the structural model to test the hypotheses which entailed examining the path-coefficients (β), t-values, p-values and R^2 generated by running a basic bootstrapping procedure with 5000 subsamples (Roky & Meriouh, 2015). Path-coefficients range from -1 to +1 with the strong positive relationships having a coefficient value closer to +1 and a negative relationship having a coefficient value closer to -1 (Hair et al., 2014). The results of the evaluations are shown in Table 5 which shows that the most significant β is between abstraction and computational thinking with a value of 0.559. However, looking at coefficients only was insufficient, and thus a two-tailed test of significance was performed using bootstrapping at a significance level of 0.05 (Hair et al., 2014). To be significant at a 5% risk of error required the paths to have t-values greater than 1.96 to be significant (Alshibly, 2014; Roky & Meriouh, 2015).

The only significant path in the initial conceptual model was abstraction with a t-value of 5.701 (p-value < 0.001). This finding supports hypothesis H1: Learning SCRATCH programming has contributed significantly to the students' abstraction (ABS) abilities. This finding resonated with the literature which considers abstraction a foundation of computational thinking (Wing, 2008; Selby & Woollard, 2014). All other hypotheses were found to be insignificant as shown in Table 5.

Table 5 - Bootstrap coefficients (β) (T) (P) (R^2)

Hypotheses	**Path**	**β**	**T-Value**	**P-Value**	**R^2**	**Decision**
H1	ABS → CT	0.559	5.701	0.000	0.601	Supported
H2	PARA → CT	0.145	0.815	0.415		Not Supported
H3	LOGIC → CT	- 0.167	1.016	0.310		Not Supported
H4	DATAR → CT	0.129	1.122	0.262		Not Supported
H5	FLC → CT	0.161	0.984	0.325		Not Supported
H6	PATG → CT	0.026	0.192	0.848		Not Supported
H7	SYS → CT	0.002	0.012	0.990		Not Supported

To assess the predictive accuracy of the model, R^2 was calculated using bootstrapping (Hair et al., 2014). Overall the conceptual model accounted for 60% ($R^{2=}$0.601) of the variance in computational thinking with abstraction having the strongest effect. As this was not deemed a satisfactory result, further analysis was conducted (Alshibly, 2014) as shown in Figure 3 below.

Path analysis was used to explore the relationship between the constructs that make up computational thinking. The aim was to identify variances in one construct that could be explained by another. To find the most reliable relationship between constructs, literal definitions of each construct as well as the construct indicator questions developed in this study were examined.

The analysis started with logical thinking (LOGIC) as it had the lowest path coefficient (-0.167). The first of three constructs that have the most notable connection with logical thinking was flow control (FLC). Flow control requires the use of logical thinking in order to plan a set of events or instructions that flow coherently (Grover & Pea, 2013; Lawanto et al., 2017). Thus, logical thinking may explain flow control. The other connections to logical thinking were parallelism (PARA) and data representation (DATAR). Parallelism is thinking along multiple channels – conceived as multitasking of thought processes - which induces logical thinking by iteratively thinking about the conditions involved in each task or channel (Grover & Pea, 2013; Brennan & Resnick, 2012; Lawanto et al., 2017; Rose et al., 2017). Data representation requires a level of logical thinking in order to show the correct order of information to solve a problem (Voogt et al., 2015; Rose et al., 2017).

Relationships between the systematic processing of information (SYS), pattern generalization (PATG) and parallelism (PARA) were also explored. Each affords explanations of the systematic processing of

information that can induce a level of pattern generalization which in turn requires the use of parallelism. The ability to understand information with deep thinking and reasoning allows the learner to link several separate pieces of information thereby generalizing data patterns (Moreno-León & Robles, 2015; Lawanto et al., 2017). These constructs could highlight sequences and constraints which use parallel thinking processes to assess similar problems and underlying patterns to form a solution for other problems (Grover & Pea, Brennan & Resnick, 2012; 2013; Djambong & Freiman, 2016; Lawanto et al., 2017; Kale et al., 2018).

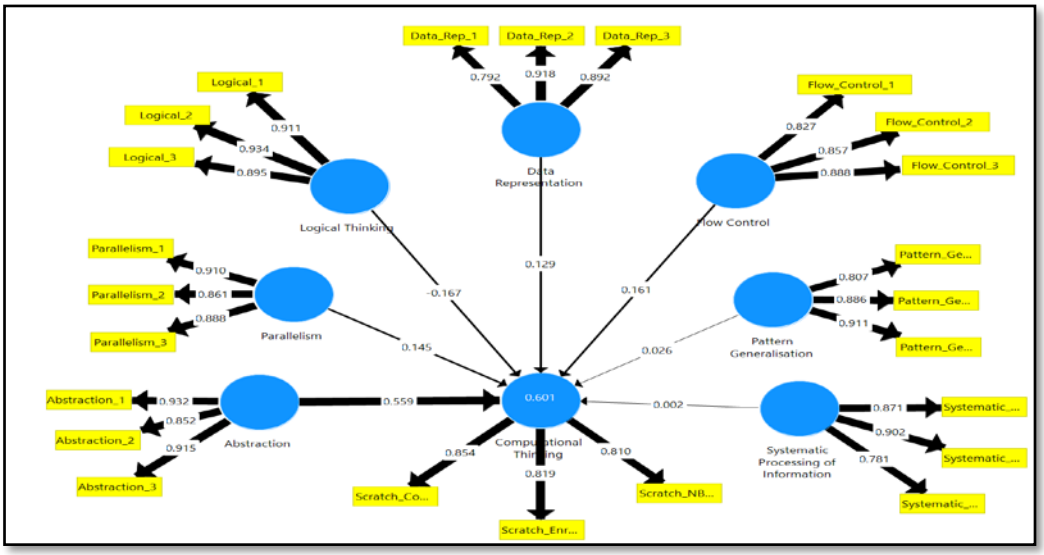

Figure 3 - PLS Path Analysis Model V1.

Finally, relationships between abstraction and data representation and logical thinking were analysed as logical thinking is required for explaining data representation through abstraction (CSTA & ISTE, 2011). To effectively reduce information to smaller parts a level of logical thinking is necessary (Brennan & Resnick, 2012; Grover & Pea, 2013; Rose et al., 2017; Lawanto et al., 2017).

Based on these relationships, six further hypotheses were derived as depicted in Table 6.

Table 6 - Cross Construct Hypotheses

Hypotheses	Description
H8	Logical Thinking (LOGIC) contributes significantly to a students' Flow Control (FLC) ability.
H9	Parallelism (PARA) contributes significantly to a students' Logical Thinking (LOGIC) ability.
H10	Pattern Generalization (PATG) contributes significantly to a students' Parallelism (PARA) ability.
H11	Systematic Processing of Information (SYS) contributes significantly to a students' Pattern Generalization (PATG) ability.
H12	Abstraction (ABS) contributes significantly to a students' Data Representation (DATAR) ability.
H13	Data Representation (DATAR) contributes significantly to a student's Logical Thinking (LOGIC) ability.

Figure 4 represents the updated conceptual model with H3: Learning SCRATCH programming has contributed significantly to the students' logical thinking (LOGIC)) ability removed from the model since it had a negative contribution with a path coefficient (β) of – 0.167.

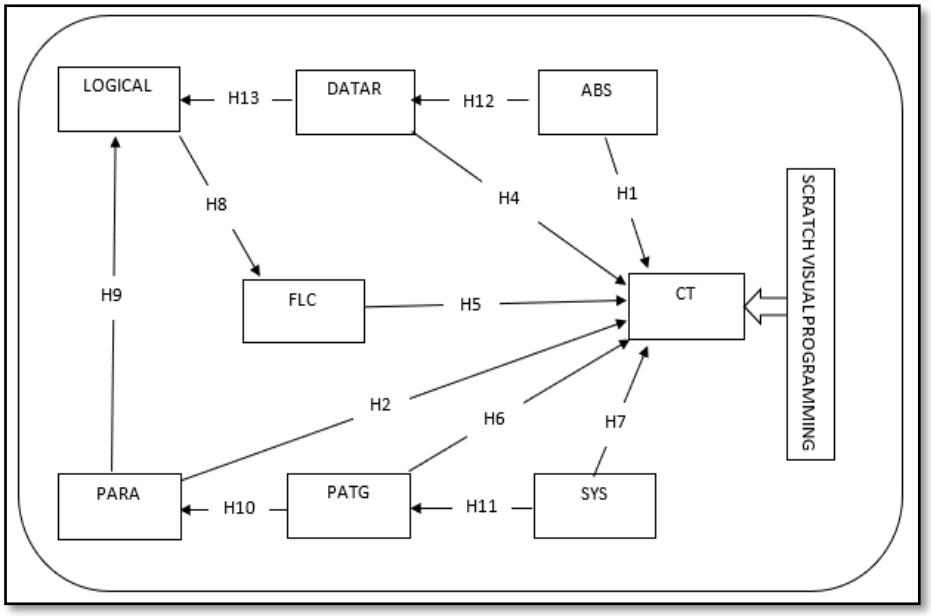

Figure 4 - Revised computational thinking conceptual constructs model.

Further analysis revealed that 7 of the 12 hypotheses were supported. The results from a two-tailed t-test with a $p<0.05$ significance level (α) showed support for the following hypotheses as can be seen in Table 7.

Table 7 - Bootstrap coefficients (β) (T) (P) (R2),

Hypotheses	Path	β	T-Value	P-Value	R^2	Decision
H1	ABS → CT	0.55	5.46	0.00	0.59	Supported
H2	PARA → CT	0.07	0.42	***0.67***		***Not Supported***
H4	DATAR → CT	0.11	0.93	***0.35***		***Not Supported***
H5	FLC → CT	0.12	0.74	***0.46***		***Not Supported***
H6	PATG → CT	0.00	0.03	***0.97***		***Not Supported***
H7	SYS → CT	0.01	0.07	***0.94***		***Not Supported***
H8	LOGIC → FLC	0.82	17.39	0.00	0.67	Supported
H9	PARA → LOGIC	0.65	6.52	0.00	0.72	Supported
H10	PATG → PARA	0.77	12.38	0.00	0.59	Supported
H11	SYS → PATG	0.76	14.11	0.00	0.57	Supported
H12	ABS → DATAR	0.57	7.29	0.00	0.32	Supported
H13	DATAR → LOGIC	0.27	2.45	0.01	0.72	Supported

To assess the predictive accuracy of the revised model, R^2 values were calculated using bootstrapping (Hair et al., 2014). The conceptual model accounted for 59% (R^2=0.59) of the variance in computational thinking with abstraction having the strongest effect. This result was similar to the earlier model. The predictive accuracy of the relationships between the constructs was explored. The results showed that 67% (R^2=0.67) of the variance in flow control was explained by logical thinking. The variance in logical thinking (72%, R^2=0.72) was explained by parallelism and data representation where parallelism had a greater path coefficient with a β value of 0.65. The results showed that 59% (R^2=0.59) of the variance in parallelism was explained by pattern generalization, while 57% (R^2=0.57) of the variance in pattern generalization was explained by systematic processing of information. Lastly, 32% (R^2=0.32) of the variance in data representation was explained by abstraction.

The relationship between logical thinking and flow control ($\beta = 0.82$, t-value = 17.39 at $\alpha = 0.05$) was observed to be significant and is supported by the flow control definition which states that a logical approach is required for flow control (Grover & Pea, 2013; Lawanto et al., 2017). The relationship with the second highest significance was between pattern generalization and systematic processing of information ($\beta = 0.76$, t-value = 14.11 at $\alpha = 0.05$) and is supported by the systematic processing of information definition which states that systematically processing information involves linking data and finding patterns (Moreno-León & Robles, 2015; Lawanto et al., 2017). The relationship between parallelism and pattern generalization follows with the third highest t-value ($\beta = 0.77$, t-value = 12.38 at $\alpha = 0.05$) indicating a significant relationship. Other significant relationships included parallelism and logical thinking ($\beta = 0.65$, t-value = 6.52 at $\alpha = 0.05$); logical thinking and data representation ($\beta = 0.27$, t-value = 2.45 at $\alpha = 0.05$) as well as data representation and abstraction ($\beta = 0.57$, t-value = 7.29 at $\alpha = 0.05$).

Discussion

Literature shows that constructs such as abstraction, parallelism, logical thinking, data representation, flow control, pattern generalization and systematic processing of information are deemed to be thinking abilities that produce computational thinking (Brennan & Resnick, 2012; Grover & Pea, 2013; Moreno-León & Robles, 2015; Voogt et al., 2015; Lawanto et al., 2017; Kale et al. 2018). The study explored and validated computational thinking definitions and constructs and presented a conceptual model developed from these constructs. The original conceptual model was revised to show the interrelated constructs and how they explain each other and presents several implications.

Computational thinking describes thought processes learnt by Non-IS students, particularly during application development and programming (Topi et al., 2010). Accordingly, research has attempted to measure the extent to which Scratch teaches computational thinking (Lai & Yang, 2011; Kalelioğlu & Gulbahar, 2014; Moreno-León & Robles, 2015). The study showed that Scratch plays a significant role in developing abstraction abilities in Non-IS majors. This finding supports literature that suggests that abstraction is the foundation of computational thinking (Wing, 2008; Selby & Woollard, 2015).

Scratch was, however, not observed to play a significant role in developing parallelism, logical thinking, data representation, flow control, pattern or systematic processing of information abilities. Nevertheless, the study does provide an indication of the construct variables and how they are interrelated. Scratch was observed to play a role in developing abstraction abilities and produces correlations between flow control and logical thinking, pattern generalization and systematic processing of information, parallelism and pattern generalization, parallelism and logical thinking, logical thinking and data representation and data representation and abstraction. Thus, Scratch teaches Non-IS majors computational thinking by inducing a level of thought processes that inherently relates to each construct. While there was limited scientific evidence of increased computational thinking, there was an increase in logical thinking which leads to flow control.

The only construct observed to directly impact computational thinking was abstraction which also impacts data representation and in turn, leads to logical thinking. Systematic processing of information leads to pattern generalization and to parallelism which impacts logical thinking. The limited observation of Scratch influencing computational thinking is borne out in the partial agreement of the respondents' perceptions of confidence in undertaking computational tasks. Although the respondents did not see learning Scratch as enriching academic life – which may be due to a lower need of logical thinking in the majority of Non-IS subjects – the majority found learning Scratch an enjoyable experience and encourage its retention in Non-IS courses.

Conclusion

The objective of this study was to explore the role of Scratch visual programming in developing the computational thinking abilities of Non-IS majors. The study emanated from a trend in literature to understand and analyze computational thinking patterns; more specifically, the ease with which visual block-based languages, such as Scratch, contribute to students learning computational thinking (Weintrop & Wilensky, 2015). Several studies have assessed visual programming languages and their

ability to foster unique problem-solving skills via computational thinking (Resnick et al., 2009; Lai & Yang, 2011; Weintrop & Wilensky, 2013) using constructs such as abstraction, parallelism, logical thinking, data representation, flow control, pattern generalization and systematic processing of information which are deemed to be thinking abilities that make up computational thinking (Brennan & Resnick, 2012; Grover & Pea, 2013; Moreno-León & Robles, 2015; Voogt et al., 2015; Lawanto et al., 2017; Kale et al. 2018). Using these constructs, this study constructed a conceptual model to explore whether Scratch facilitates the development of computational thinking in Non-IS students. The results of a survey indicated that Scratch played a significant role in abstraction for developing computational thinking. Further analysis of the relationships between computational thinking constructs concluded that Scratch played a role in developing logical thinking by acting through abstraction and the constructs of data representation, systematic processing of information, pattern generalization, parallelism, and leading to flow control. Nevertheless, these were not observed to significantly influence computation thinking.

Further research is required to link logical thinking to computational thinking. Furthermore, additional research is required to determine if flow control has a mediating or moderating impact on computational thinking. Time was a key limitation of this study. Firstly, the Scratch course lasted only 7-weeks which may be too short to significantly influence computational thinking in Non-IS majors. Secondly, since the length of elapsed time between the respondents completing the 7-week Scratch course and the study varied from one to five years, perceptions of the respondents may have changed over the intervening periods.

References

Alshibly, H. (2014). Evaluating E-HRM success: A validation of the information systems success model. *International Journal of Human Resource Studies*, (4:3), pp.107-124.

Brennan, K., & Resnick, M. (2012). Using artefact-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. pp.1-25.

Djambong, T., & Freiman, V. (2016). *Task-Based Assessment of Students' Computational Thinking Skills Developed through Visual Programming or Tangible Coding Environments*. International Association for Development of the Information Society.

Grover, S. & Pea, R. (2013). Computational thinking in K—12: A review of the state of the field. *Educational Researcher*, (42:1), pp.38-43.

Hamid, M. R., Sami, W., & Sidek, M. M. (2017). Discriminant Validity Assessment: Use of Fornell & Larcker criterion versus HTMT Criterion. *Journal of Physics*: Conference Series, (890:1).

Hair, J. F., Ringle, C. M., & Sarstedt, M. (2013). Partial least squares structural equation modelling: Rigorous applications, better results and higher acceptance. *Long Range Planning*, pp.1-12.

Hair, J. F., Sarstedt, M., Hopkins, L., & G. Kuppelwieser, V. (2014). Partial least squares structural equation modelling (PLS-SEM) An emerging tool in business research. *European Business Review*, (26:2), pp.106-121.

Hair, J. F., Hult, G. T. M., Ringle, C., & Sarstedt, M. (2016). *A primer on partial least squares structural equation modelling (PLS-SEM).* Sage Publications.

Israel, M., Pearson, J., Tapia, T., Werfel, Q., & Reese, G. (2015). Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. *Computers & Education*, (82), pp.263-279.

CSTA and ISTE. (2011). *Computational Thinking Leadership Toolkit*. Retrieved from http://www.iste.org/docs/ctdocuments/ct-leadershipt-toolkit.pdf?sfvrsn¼4.

Lai, A. F., & Yang, S. M. (2011). The learning effect of visualized programming learning on 6th graders' problem solving and logical reasoning abilities. In *Electrical and Control Engineering (ICECE), 2011 International Conference*, pp.6940-6944.

Lawanto K., Close K., Ames C., Brasiel S. (2017) Exploring Strengths and Weaknesses in Middle School Students' Computational Thinking in Scratch. In: Rich P., Hodges C. (eds) *Emerging Research, Practice, and Policy on Computational Thinking. Educational Communications and Technology: Issues and Innovations*. Springer, Cham.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behaviour*, (41), pp.51-61.

Kalelioğlu, F., & Gülbahar, Y. (2014). The effects of teaching programming via scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education - an International Journal*, 13(1), 33-50.

Kalelioğlu, F., Gülbahar, Y., & Kukul, V. (2016). A framework for computational thinking based on a systematic research review. *Baltic Journal of Modern Computing*, (4:3), pp.583.

Moreno-León, J., & Robles, G. (2015). Dr scratch: A web tool to automatically evaluate scratch projects. Proceedings of the *Workshop in Primary and Secondary Computing Education*.132-133.

Moreno-León, J., & Robles, G. (2016). Code to learn with Scratch? A systematic literature review. *Global Engineering Education Conference (EDUCON)* IEEE, 150-156.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., & Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM*, (52:11), pp.60-67.

Roky, H., & Al Meriouh, Y. (2015). Evaluation by users of an industrial information system (XPPS) based on the DeLone and McLean model for IS success. *Procedia Economics and Finance*, (26), pp.903-913.

Rose, S., Habgood, J., & Jay, T. (2017). An exploration of the role of visual programming tools in the development of young children's computational thinking. *Electronic Journal of e-Learning*, (15:4), pp.297-309.

Saez-Lopez, J., Gonzalez, M., & Vazquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two-year case study using "Scratch" in five schools. *Computers & Education*, (97), pp.129-141.

Selby, C., & Woollard, J. (2014). *Refining an understanding of computational thinking*. Retrieved from https://eprints.soton.ac.uk/372410/1/372410UnderstdCT.pdf

Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies*, (18:2), pp.351-380.

Su, A. Y. S., Huang, C. S. J., Yang, S. J. H., Ding, T. J., & Hsieh, Y. Z. (2015). Effects of Annotations and Homework on Learning Achievement: An Empirical Study of Scratch Programming Pedagogy. *Educational Technology & Society*, (18:4), pp.331–343.

Topi, H., Valacich, J. S., Wright, R. T., Kaiser, K., Nunamaker Jr, J. F., Sipior, J. C. & Jan de Vreede, G. (2010). IS 2010: Curriculum guidelines for undergraduate degree programs in information systems. *Communications of the Association for Information Systems*, (26), p.59.

Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, (20:4), pp.715-728.

Weintrop, D., & Wilensky, U. (2013). Supporting computational expression: How novices use programming primitives in achieving a computational goal. Paper presented at the *AERA*, at San Francisco, CA.

Weintrop, D., & Wilensky, U. (2015). To block or not to block, that is the question: Students' perceptions of blocks-based programming. Proceedings of the *14th International Conference on Interaction Design and Children*. pp.199–208. DOI: http://doi.acm.org/10.1145/2771839.2771860

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

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society: Mathematical, Physical and Engineering Sciences*, 366 (1881).

Wing, J. M. (2011). *Computational thinking: what and why? The Link*. Retrieved from https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why