An investigation into pre-service teachers' experiences while transitioning from Scratch programming to procedural programming

[a]Fatimah Tijani; [b]Ronel Callaghan, and [b]Rian de Villers

[a]Michael Otedola College of Primary Education, Lagos, Nigeria;*
[b]Faculty of Education, University of Pretoria, South Africa
[*]Corresponding author. Email: tijanifatima.tf@gmail.com

Abstract

The use of Scratch programming in introducing text-based programming to novices at all levels of education has gained prominence in computer science but is still hardly known among pre-service teachers. With affordances of Scratch in learning text-based programming, we present an experience report on how we supported our first-year pre-service teachers' learning of procedural programming concepts with Scratch for the first time. The study follows an action research strategy conducted over two cycles with 58 pre-service teachers who were purposively sampled. Findings revealed that Scratch supported the learning of procedural programming by our first-year pre-service teachers to some extent. We, therefore, recommend that pre-service teachers be exposed to more exercises while focusing on challenging concepts such as algorithms, use of variables, repetition, and control structures.

Keywords: Action research, pre-service teachers, procedural programming, teaching and learning process framework (TLPF), Scratch programming

Introduction

Across Nigeria, colleges of education have been established with the purpose of training and preparing students to become teachers who will teach at primary and junior secondary schools (Aina, 2015). Adequate preparation of pre-service teachers studying computer science and programming related courses is necessary for technology innovation at all tiers of education in the country. Surprisingly, many pre-service teachers fail and lose interest in programming, which is one of the basic skills for computer science teachers in Nigeria (Olelewe & Agomuo, 2016). Pre-service teachers in this context lack programming experience. They are also unable to understand basic programming concepts and write simple programming codes. These issues impede achieving the goals of teaching and learning computer programming in Nigeria (Olelewe & Agomuo, 2016). Some of the factors contributing to student difficulties in programming include the complex nature of programming teaching (Koulouri et al., 2015); the idiosyncratic nature and complex syntax of programming (Topalli & Cagiltay, 2018); problem-solving skills (Yurdugül & Aşkar, 2013); and traditional teacher-centred teaching which does not focus on student's intelligence and learning capabilities (Olelewe & Agomuo, 2016). Taking these factors together, students lose interest in the course, resulting in truancy, learning difficulties, student failure, and often high dropout rates in programming (Law et al., 2010). Therefore, if pre-service teachers are unable to understand the skills of programming, it will be difficult to prepare young learners at the primary and secondary schools to meet the goals of tertiary education as stated in the National Policy on Education (Federal Government of Nigeria, 2013).

To address some of these identified issues, one of the promising approaches that have been suggested is the use of visual programming languages (VPL) as a support for the learning of text-based programming. It is speculated that if programming courses are supported with VPL, students may understand programming concepts better and thereby develop an interest in programming. However, there is a dearth of literature on the teaching of procedural

programming (PP) with the VPLs among pre-service teacher education. With this in mind, we promulgate the importance of introducing visual programming (VP) into the computer science curriculum and studying pre-service teachers' experiences as they transition to PP.

Previously, the teaching of programming was mostly delivered in a traditional environment. The development of programming skills cannot be facilitated through rote learning as is done in the traditional classroom, but through the construction of knowledge within a social environment. For this reason, it is necessary to design instruction that supports student-centred teaching and learning of programming. It is conjectured that introducing VP within a social environment will help our first-year students at tertiary institutions to learn to program better. This paper presents the results of our investigation.

The research question guiding this study was: How does Scratch programming support improved learning of procedural programming among first-year pre-service teachers?

Literature Review

The teaching of introductory programming to novices through exposure to VP, such as Scratch, before transitioning to text-based programming is important if novices are to become fluent programmers in the future (Shapiro & Ahrens, 2016). Researchers and practitioners in computer science education have argued the benefits of Scratch, heralding it as being different to text-based programming. The benefits documented include an improvement in students' competence on loops and conditionals when Scratch is used with Logo (Lewis, 2010); a positive influence with Java (Malan & Leitner, 2007); an improved attitude towards programming (Mladenović et al., 2016); high cognitive levels; increased motivation and self-efficacy with Java or C++ (Armoni et al., 2015); and perceived easiness with Java (Weintrop & Wilensky, 2015). In contrast, Martínez-Valdés et al., (2017) as well as Marimuthu and Govender (2018) stressed that Scratch was less satisfactory when used as a precursor to Java and that students frowned at an informal introduction of Scratch. Other studies have also supported Scratch as showing promising results on the affective aspects, but concepts such as variables, concurrency, and repeated execution could not be internalised by the students (Meerbaum-Salant et al., 2013). Nevertheless, these concepts could be grasped through improved instruction.

However, despite the affordances of Scratch as a precursor to text-based programming, researchers are still uncertain about its long-term benefits. Some critiques include its lack of authenticity; less powerful technique and long-winded blocks (Weintrop & Wilensky, 2015); lack of mediated transfer (Krpan et al., 2017); students developing bad habits; and use of a bottom-up approach to programming (Moreno & Robles, 2014). Learners may exhibit these shortcomings while programming. It is, therefore, necessary for the instructors to make use of the affordances of Scratch while noting its weaknesses, and fully prepare students in text-based programming. Meerbaum-Salant et al., (2011), therefore, recommended that teachers must focus the teaching of programming on algorithm design and complex structures which will help learners to code at a higher level. However, transitioning from Scratch to other text-based programming languages requires 21st-century skills. Using the right teaching approach that will foster this, as suggested by Resnick et al., (2009), requires a teaching approach that involves a combination of diverse project types, personalisation of Scratch projects, and social collaboration. Meerbaum-Salant et al., (2011) also stressed the use of a constructivist teaching approach with a focus on exploration and experiment.

Theoretical Underpinnings

Procedural Text-based Programming Languages

PP provides varying commands for structuring and manipulation of codes (Vujošević-Janičić & Tošić, 2008), and allows the programmer to state the computations that change the program code using procedures (Lindeman et al., 2011). QBASIC is a type of PP. It has a user-friendly environment, is easier to use, portable, and has application packages suitable for programming for first-year students. QBASIC, therefore, is the first introductory programming course for students at the universities and colleges of education in Nigeria.

Visual Programming Languages

Visual programming languages (VPL) support the use of a graphical user interface with each programming example displayed using graphical objects (Aleksic & Ivanovic, 2016). It is argued that novices will find VP easier to use and better than PP because of its ability to support forward and backward reasoning, activate memory, and present a visual representation of the control and data flow in a program (Lye & Koh, 2014). The cognitive load found in PP is reduced by the chunking of codes into smaller units by helping learners focus on the codes rather than the syntax of the program (Cetin, 2016). Scratch is a type of VPL; it was developed with the notion that it will lower barriers to learning programming by empowering novices to master programming constructs and logic before learning real programming. Although Scratch is specifically designed for primary or secondary school learners, it is used in this study because learners in this context lack previous programming experience.

Constructivism and Constructionism

Constructivism as a theory of learning is informed by the work of Piaget and Vygotsky. While Piaget believes learning is developmental and involves the mental construction of knowledge, Vygotsky believes that socially constructed knowledge facilitates human development (Schunk, 2014). For Piaget, construction of knowledge is personal, but for Vygotsky, shared knowledge facilitates construction of knowledge. The two major perspectives of constructivism are cognitive and social constructivism. Each perspective holds different assumptions about learning. Assumptions of the social constructivism of Vygotsky are that *social interaction* forms the basis of construction – individual learners become *self-regulated* as internalisation is formed through mental constructions that evolve during social interaction. *Language* is a vital tool in social interaction and the *zone of proximal development* (ZPD) encourages cognitive development. Strategies for implementing constructivism in the teaching of programming include instructional scaffolding, pair programming (Chetty & Barlow-Jones, 2014), and cooperative learning (Johnson & Johnson, 2009).

Constructionism means "learning by making" (Papert & Harel, 1991, p. 6). It focuses on the art of learning through building or sharing designed objects (Girvan et al., 2013). Constructionism shares the same worldview on learning as constructivism, although there is a slight difference. On the one hand, the former focuses on learning at every stage of development and the learner's construction of knowledge. n the other hand, the latter deals with consciously engaging a learner with the creation and modification of digital artefacts.

Methodology

Research Strategy and Paradigm

This study employed an action research (AR) strategy. An AR is systemic and studies a problem by developing theories to effect change. The researchers used two cycles of practical AR to study an identified problem in their own classroom to improve professional practice. The study was guided by an interpretive paradigm and hermeneutic phenomenology as a method of inquiry.

Population and Sample
The population comprised first-year computer science students enrolled in the 2015/2016 and 2016/2017 academic sessions. More specifically, these students were exposed to QBASIC – a compulsory introductory programming course – in their first semester in college. Purposive sampling was used to obtain a sample of 58 students in the two academic sessions.

Setting and Participants
The setting of the study was a college of education situated in a rural area of Lagos State, Nigeria. The participants comprised first-year pre-service teachers who were enrolled to study computer science. Out of the 58 pre-service teachers, only four (4) had previous training on the theoretical aspect of QBASIC programming, the rest had never learned to program in secondary school. All the students were new to Scratch. In the two cycles, one of the researchers acted as both a participant observer and human instrument for data collection.

Research Procedure
This study made use of Du Toit's (2010) visionary AR model, which comprises the following five stages, namely: (1) planning for innovation, (2) acting to innovate, (3) observing the effects of the new action, (4) reflecting in/on the action, and (5) evaluating. The five stages were followed in the two AR cycles. In the *planning phase*, we reviewed the literature, and the best practices from the literature were brought into the study. This informed the design of a teaching and learning process framework (TLPF) prepared by the authors, as illustrated in Figure 1 below.

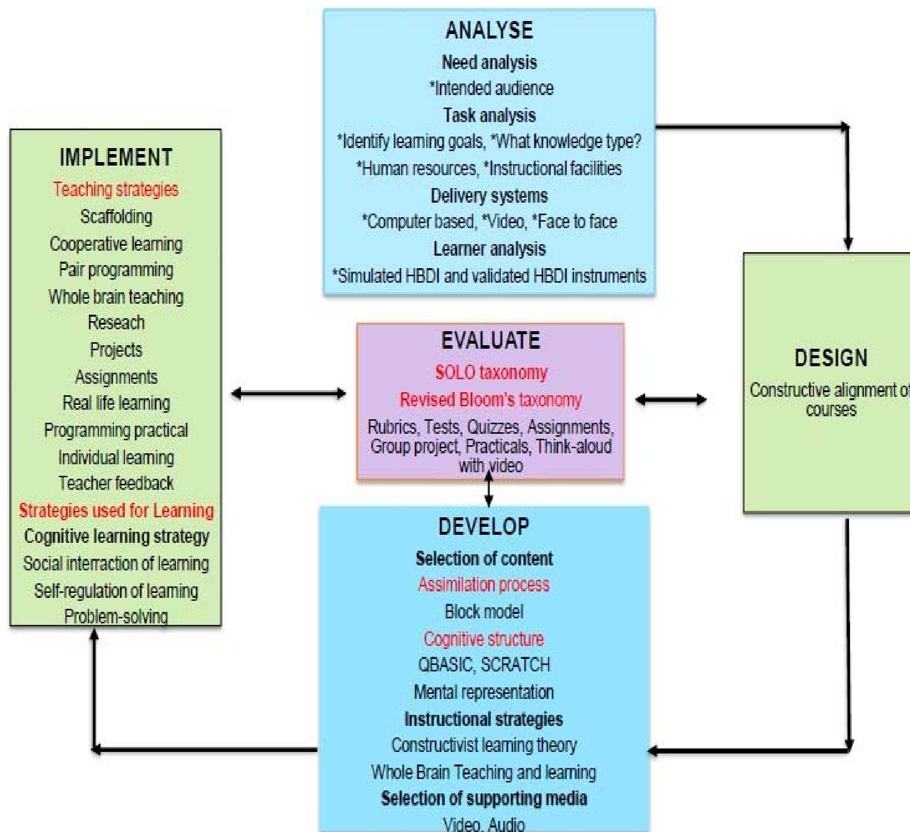

Figure 1: Teaching and Learning Process Framework

The TLPF was based on the ADDIE – analyse, design, develop, implement, and evaluation –

framework for instructional design. In addition to being simple to use, ADDIE provides instructors with a systematic approach in designing and developing a learning experience, with the outcome of each phase informing the other (Khalil & Elkhinder, 2016). Therefore, this framework was considered suitable for this study because it is system-oriented and produces a good instructional design. The underlying motivation for the design of the TLPF was to teach students programming using a student-centred and holistic approach to learning. The processes involved in the planning of the framework are described below.

The planning of the TLPF

Analysis: Pre-service teachers' approaches to learning were first determined and further guided instructional design. This aspect is not discussed in this paper.

Design: Constructive alignment of courses (Scratch and QBASIC) were designed by the researchers, and this guided instruction and assessment planning (Biggs, 2012).

Development: The designed instruction formed the initial TLPF. The content and supporting media were organised to achieve the objective and create a satisfying learning experience for the students. Different methods were used to communicate the content (Scratch and QBASIC), which was based on the constructivist theory of learning.

Implementation: Instructional activities were planned and facilitated using constructivist principles such as pair programming, scaffolding, and cooperative learning towards the achievement of the designed learning objectives. Instructional activities necessitated grouping, which strengthened the collaboration among pre-service teachers and the researchers. Opportunities for the construction of meaning from the learning experiences were encouraged.

Evaluation: Assessment tasks were designed based on the structure of the observed teaching and learning framework (SOLO) and revised Bloom's taxonomy (Biggs, 2003; Biggs & Tang, 2007). Different assessments, such as assignments, group projects, and presentations were also considered during the planning phase.

The Acting Phase of the TLPF

In the *acting phase*, the instruction was facilitated using the prepared TLPF. In the first cycle, participants were introduced to Scratch over a four-week period, followed by QBASIC. Classroom practicals, assignments, group work, and project works were done individually and cooperatively during programming lessons. Participants received scaffolding where needed and were also exposed to programming assessments at different stages of the teaching sessions. Assessments included classroom exercises, a test of individual concepts, interim tests 1-4, and a final test.

The Observation Phase

During the *observation phase*, the collected data was reflected upon. The insights obtained during this phase were used to inform the teaching decisions that supported students' learning in each class. The outcomes of the first cycle formed the basis for re-planning in the second cycle. As we reflected on the data, we also questioned our decisions to think differently and engaged in an ongoing interim analysis as data collection unfolded. We reflected on the data to check how it informed and captured the instruction. We finally *evaluated* the teaching situation to ascertain whether we lived our values brought into the study. The outcome of the first cycle of AR was used to enhance the redesign of the $TLPF_1$, which was used in the second cycle as $TLPF_2$ (see Figure 1). The second cycle started with a re-planning for innovation, which was informed by the outcome of the first cycle, including the management of time, increased

programming and lecture time, as well as focus on concepts such as variables, flowcharts, and algorithms. These outcomes were incorporated into the second cycle which necessitated the teaching of Scratch and QBASIC on separate days of the week. This helped the participants to explore each environment extensively in the second cycle. The participants in this cycle differed from the participants in the first cycle.

Data Collection Methods

Data was collected through classroom observation, artefacts, interviews, and documents. The classroom observations took place from 20 January – 7 April 2016 with 11 lessons observed in the first cycle; and 27 January – 7 April 2017 with 21 lessons observed in the second cycle. Each lesson was observed using video and structured observation, which focused on the participants' construction of understanding, social interactions, shared meanings, and their behaviour as lived in the programming classroom. The artefacts comprised classroom assessments and a final test which covered all topics learnt during the semester, including algorithms, data types, variables, repetition, looping, and program writing. An interview protocol was used to elicit responses from 14 participants who had a lived experience in both Scratch and QBASIC, and each audio-recorded interview lasted approximately 45 minutes. The participants also jotted down their reflections of their experience in the programming classroom in their journal, and these were studied for further analysis.

Data Analysis

The hermeneutic cycle was employed for data analysis (Klein & Myers, 1999). After the interviews were transcribed and re-read, they were analysed both deductively and inductively using thematic analysis. All the data was coded by the first author and then checked by the co-authors for correctness.

Quality Assurance and Ethical Aspects

Validity was ensured by giving the participants an opportunity to respond during the interview session, obtaining rich qualitative data and verifying participants' responses, and discussing the results and findings with a critical friend. *Trustworthiness* was maintained through the use of triangulated data, member checking, keeping an open mind during the interview, thick description of the phenomenon under study, negative case analysis, prolonged time in the field, and ongoing reflective analysis (Creswell, 2014). All ethical considerations were observed and permission to conduct the study was granted by the college management of the said institution. In addition, permission was given to withdraw from the interview at any time and anonymity was achieved through the use of pseudonyms. (Thus, the names – Nancy, Vanessa, Mercy, Uchenna, Farai, Peter, and Ramsey – are pseudonyms used to protect the identities of the participants).

Findings

The two sub-themes described below explain the emerged theme – programming knowledge gained by students.

Students' Programming Knowledge

We used the quantitative data presented below to gain a better understanding of the qualitative findings generated from the data. Participants were given similar exercises on programming concepts for both Scratch and QBASIC programming. Table 1 gives a summary of the percentage of correct answers of all programming activities students did in both the first and second cycle. Findings revealed that students in the first cycle improved on variables, expressions, and program writing with a low percentage score on algorithms, operators, and repetition, while in the second cycle, exposure to Scratch programming for the whole semester

did not seem to enhance their understanding of QBASIC concepts. However, it is possible that further investigation of the final tests might provide better results.

Table 1. Programming concepts assessed during classroom activities

Programming concepts	1st cycle		2nd cycle	
	Scratch scores (%)	QBASIC scores (%)	Scratch scores (%)	QBASIC scores (%)
Variables	39	98	60	55
Debugging	Not tested	81	Not tested	50
Algorithms	79	37	47	27
Operators	50	45	58	32
Repetition	56	50	61	26
Expressions	47	60	58	33
Program writing	18	73	46	42

Table 2 presents the final assessment test for defining a variable, debugging of syntactic errors, designing algorithms for solutions, expressions and repetitions, and an analysis of the question types. The final test for the cycles was administered using pen and paper with 18 and 22 students present in the two cycles, respectively. The test, which lasted for 1 hour, covered all topics learnt over the semester.

Table 2. Final Test on Programming Concepts in Both Cycles

Question Type	Programming concepts learned	Revised Bloom's	SOLO	% Correct Answers (1st cycle)	% Correct Answers (2nd Cycle)
(1) Basics	Data types	Remember	Unistructural	56%	56.4%
(2) Syntactic errors	Debugging	Remember	Unistructural	16%	73%
(3a) Skeleton code	Program writing	Apply	Relational	16.5%	70%
(3b) Code tracing	Expression and operators	Apply	Unistructural	33%	28%
(4ai) Code tracing	Variables	Remember	Unistructural	43%	36%
(4aii) Code purpose	Explaining skill	Understand	Relational	33%	68%
(4b) Change in representation	Algorithm	Understand	Relational	Not tested	36%
(5ai) Code tracing	Data types	Analyse	Unistructural	19%	38%
(6b) Syntactic error	Debugging	Understand	Unistructural	Not tested	38%
(6c) Change in representation	Program writing	Evaluate	Relational	12%	14%
(7a) Change in representation	Program writing	Create	Relational	61%	26%
(7b) Syntactic errors	Debugging	Understand	Relational	35%	35%

Table 2 shows that participants in both cycles experienced difficulty with some programming concepts. For example, on data types involving code tracing and analysis (5ai), the percentage correct answers was 19% and 38%, respectively, but the students had an average performance on data types (1) that required them only to recall facts. On debugging (2, 6b, and 7b), the percentage correct answers for students in the first cycle was very low, but in the second cycle, the result showed that the students understood debugging to some extent with their understanding below average as the levels of the questions on the taxonomy became higher. Findings on expressions and operators (3b) showed that the students' understanding was low, and when compared to the results in Scratch, it showed they were only at the average in both cycles with the second cycle maintaining a lower percentage. On program writing (3a, 6c, 7a), the first cycle students' percentage score was low at the lower levels of Bloom, but higher as it

moved to the top level. The reverse was the case in the second cycle. On variables (4ai), the result showed that the percentage score was low in both cycles. However, on algorithms (4b), the percentage score in the final test is not shown for the first cycle, but in the classroom activities, the result was low in the first and second cycle.

Students' Perspectives about Scratch and QBASIC

Some participants perceived Scratch as a foundation stage that every student must master. This understanding was fuelled by friends and seniors who were exposed to Scratch. The following excerpts from the interview and reflective learning journal support this finding:

> "*Scratch is just like a foundation …*" (Ramsey).

> "*…when most of them learnt that QBASIC is our real course, why then are we doing Scratch… but it has helped me*" (Percy).

> *"QBASIC programming not so very basic for me. More like complex programming"* (Percy).

Knowledge Gained in Scratch and QBASIC

This category describes pre-service teachers' lived experiences of concepts such as algorithms with flowchart and pseudocode, variables, repetition, control structures, and expressions. All the participants noted that their understanding of programming concepts was well-grounded due to repeated teaching combined with individual and group practices in both programming languages. An interview excerpt from Farai supports this finding:

> "*…so being taught something repeatedly, you will get more understanding about the topic".*

However, one participant – Uchenna – expressed difficulty with the learning of programming due to the mathematical aspect.

- Algorithms

Learning *algorithms* with flowchart and pseudocode was an interesting topic for half of the participants. When designing a solution to problems, the knowledge gained from the algorithm steps in the Scratch class were applied to solving problems in QBASIC. This was achieved as the students noted that most of the topics learnt in class, including algorithms, have been learnt previously (up to three times) in the Scratch class. Thus, learning it again in the QBASIC class increased their understanding of the concepts. However, classroom observation shows that students do not always apply the algorithmic steps during problem-solving in QBASIC, evident as follows:

> *"…students don't know when to use a decision in a program and did not apply algorithmic techniques in writing program…"* (Researcher 1).

To support this further, classroom assessments on algorithms in Tables 1 and 2 on the final test was 36% for the second cycle. Further investigation gives a deeper understanding of this. Nancy, Vanessa, Mercy, and Ramsey indicated in their reflective learning journal that they struggled with learning the concepts of algorithms during their first time in class. For example, Ramsey said:

> *"…difference between algorithms and a flowchart, because it was the first class so I couldn't understand better…"*

The first class here might refer to the different times the students joined the class. Not being part of the class on the first day classes began, where foundation topics were introduced and explained, may have contributed to this.

- Variables

Students in the first cycle can correctly identify and define a variable for programs in QBASIC. Although the variable naming convention from Scratch either uses or does not use spaces between variable names, it does not affect how students name variables in QBASIC. It is uncertain whether Scratch helped them, since they had a low percentage score in QBASIC (see Table 1). However, there seems to be an improvement in QBASIC for students in the second cycle. A further investigation from the second cycle participants showed that even though they were taught variables and its application in solving programming problems, and performed better than their first cycle counterparts, most of the participants did not always declare variables for programming problems. Data from three classroom observations showed that,

> *"[G]roups (C, D, E, F and G) also presented their results too without making use of variables in their solution… I noticed the students did not all make use [of] variables in their group work" and "in the problem given, some of them used the 'number of hours' without declaring the number of hours as a variable"* (Researcher 1).

Results from Table 2 on the final test also support the findings.

- Repetition and control structure

Participants noted that the concepts of "repetition" and "control structure" learnt in Scratch are also found in QBASIC. Therefore, transitioning from Scratch to QBASIC with several activities on repetition animations in Scratch using the repetition structures aided understanding of related commands in QBASIC. A classroom observation supports this finding,

> "*…they all did well in the practical…"* (Researcher 1).

In addition, results from Table 2 indicate that they performed above average on repetition. However, not all the students have an understanding of repetition structure and writing program codes involving "control structure". This could be because the concepts were introduced towards the end of the semester with little practical opportunities. For example,

> *"I understood the control structure, but I don't know how to write and solve some questions under it"* (Vanessa).

Results from Table 1 show that students in the first cycle have an above-average percentage score in both languages, but a low percentage score for QBASIC in the second cycle.

- Debugging skills

Furthermore, Scratch contributed to developing *debugging skills* in programming. Even though debugging was not taught as a topic in Scratch, participants affirmed that debugging was not an issue in Scratch class because Scratch did not return their errors. However, they came to the awareness that locking the wrong blocks does not always produce the desired result. But through further attempts based on trial and error, the expected result was obtained, even though they didn't understand the algorithm of the program. A supporting finding during a classroom

teaching where students were asked to discuss what they learned while programming is explained in the following statement by Peter:

> "*…we made a mistake where we are trying to make a variable for dragon-1; we write it without knowing that we are going to select it inside a block. So, when we run the program, we discovered that after we inserted a cough, the dragon does not respond*".

Results from Tables 1 and 2 further show that participants' preconceived ideas about debugging with a lack of understanding concerning the algorithm of the program might have contributed to the low percentage of correct answers in questions 2, 6b, and 7b.

- Program writing

The participants claimed that program writing as a result of block arrangements in Scratch contributed to the arrangement of program codes in QBASIC. This was revealed in their ability to develop a QBASIC flowchart and pseudocode from a Scratch script (see Table 1). As such, they could understand that wrongly arranged blocks in Scratch give wrong output, meaning program codes that are not well-arranged produces wrong output in QBASIC. This was affirmed by Peter as follows:

> *"… in Scratch, there are some programs that when you place it in the wrong position, it will stop there. Like when you are supposed to put something in a LOOP… and you don't put it… the output will be very wrong. So, in QBASIC also, if you are to face a problem and you do not arrange it in order, then you will not get what you want to get".*

However, program writing is more than just snapping blocks of codes together. Further investigation of Scratch support to QBASIC in this regard is not clear, as there was a low performance on Scratch and higher performance on QBASIC in the first cycle, with a slight difference in the second cycle (see Table 1). As discussed earlier, students do not apply the use of variables during program writing in QBASIC. It was towards the end of the semester that students started seeing the importance of variables for planning solutions and writing programs in QBASIC. This, however, was not the case during Scratch programming. Participants always define variables using the "set block". The reason for this could be that most classroom exercises students were exposed to in Scratch were based on pre-written examples from the textbook in which students only remix to produce a new script and animation.

Discussion

This study investigated pre-service teachers' experiences as they transitioned from Scratch to procedural programming. Pre-service teachers saw Scratch as easy to learn, motivating their interest in programming (Ouahbi et al., 2015). They also appreciated the benefits of Scratch for the learning of programming because it served as a foundation for learning QBASIC. However, they believed it did not allow them to express themselves well during program writing (Weintrop & Wilensky, 2015). Knowledge gained in Scratch deepened their understanding of concepts like algorithms, repetition, and variables in QBASIC. The pre-service teachers also learned program writing through the arrangement of program blocks in Scratch, which was mostly based on trial and error. This finding on program arrangement relates to "sequencing". Bers et al., (2014) explained sequencing as a form of planning involved when arranging computer codes to achieve the desired result. Therefore, students transferred the knowledge to the arranging of codes in QBASIC. However, they could not think algorithmically about programs that involved complex structures like repetitions and control structures in QBASIC (Grover & Basu, 2017; Moreno & Robles, 2014). What we learned from this study is that

Scratch supports the learning of procedural programming in our first-year pre-service teachers to some extent. We believe that exposure to more exercises while focusing on challenging concepts such as algorithms, appropriate use of variables, debugging, repetition and control structures, will deepen their understanding of these concepts.

Conclusion

Supporting the learning of QBASIC with Scratch programming motivated pre-service teachers to learn to program. It also facilitated their understanding of the relevant concepts. However, doing complex exercises was problematic for pre-service teachers. Thus, in the future, we would like to focus more on extensive practicals on algorithms, debugging, and repetition and control structure by exposing pre-service teachers to more exercises that will enable them to think algorithmically. There is an urgent move by the Nigerian government to develop coding skills in learners at the primary and secondary school level using VPLs. Its implementation can only be effective if pre-service teachers are well prepared. In the interim, therefore, we recommend the use of Scratch as the first programming language for pre-service teachers in their first year of college to equip them not only for QBASIC but for other programming courses that will be done before graduation. With the introduction of Scratch, they will be motivated to learn to program and thereby develop programming skills. We also recommend the use of TLPF for programming teachers and lecturers, which they can adapt or adopt for designing student-centered programming instructions.

References

Aina, J. K. (2015). Analysis of integrated science and computer science students' academic performances in Physics in colleges of education, Nigeria. *International Journal of Education and Practice, 3*(1), 28-35.

Aleksic, V., & Ivanovic, M. (2016). Introductory programming subject in European higher education. *Informatics in Education, 15*(2), 163-182.

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to "Real" programming. *ACM Transactions on Computing Education, 14*(4), 137-151.

Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers and Education, 72*, 145-157.

Biggs, J. (2003). Aligning teaching and assessing to course objectives. In M. L. Sein-Echaluce, A. Fidalgo-Blanco, & F. J. Garcia-Peñalvo (Eds.), *Teaching and Learning in Higher Education: New Trends and Innovations* (pp. 13-17). University of Aveiro.

Biggs, J. (2012). Enhancing learning through constructive alignment. In J. R. Kirby & M. J. Lawson (Eds.), *Enhancing the quality of learning: Dispositions, instruction, and learning processes* (pp. 117-136). Cambridge University Press.

Biggs, J., & Tang, C. (2007). *Teaching for quality learning at University* (3rd ed.). Open University Press.

Cetin, I. (2016). Pre-service teachers' introduction to Computing: Exploring utililization of Scratch. *Journal of Educational Computing Research, 54*(7), 997-1021.

Chetty, J., & Barlow-Jones, G. (2014). Novice students and computer programming: Toward constructivist pedagogy. Mediterranean Journal of Social Sciences, 5(14), 240. doi: 10.5901/mjss.2014.v5n14p240

Creswell, J. W. (2014). *Research Design: quantitative, qualitative and mixed methods approaches* (4th ed.). Pearson Education Inc.

Du Toit, P. (2010). *An Action Research approach for monitoring one's professional development as a manager*. Foundation for professional development.

Federal Government of Nigeria. (2013). *National Policy on Education*. NERDC Press.

Girvan, C., Tangney, B., & Savage, T. (2013). SLurtles: supporting constructionist learning in

second life. *Computers and Education, 61*, 115-132.
Grover, S., & Basu, S. (2017, 8–11 March). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. *Paper presented at the Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education*, Seattle, WA, USA.
Johnson, D. W., & Johnson, R. T. (2009). An educational psychology success story: Social interdependence theory and cooperative learning. *Educational Researcher, 38*(5), 365-379.
Khalil, M. K., & Elkhider, I. A. (2016). Applying learning theories and instructional design models for effective instruction. *Advances in Physiology Education, 40*, 147-156.
Klein, H. K., & Myers, M. D. (1999). A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Quarterly, 23*(1), 67-93.
Koulouri, T., Lauria, S., & Macredie, R. D. (2015). Teaching introductory programming: a quantitative evaluation of different approaches. *ACM Transactions on Computing Education, 14*(4), 1-28.
Krpan, D., Mladenović, S., & Zaharija, G. (2017, 22–26 May). *Mediated transfer from visual to high-level programming language*, Opatija, Croatia.
Law, K. M., Lee, V. C., & Yu, Y.-T. (2010). Learning motivation in e-learning facilitated computer programming courses. *Computers and Education, 55*(1), 218-228.
Lewis, C. M. (2010, March 10–13). How programming environment shapes perception, learning and goals: logo vs. scratch. In *Proceedings of the 41st Technical Symposium on Computer Science Education (pp 346-350).*
Lindeman, R. T., Kats, L. C., & Visser, E. (2011, 22–24 October). Declaratively defining domain-specific language debuggers. *Paper presented at the ACM SIGPLAN Notices,* Oregon.
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 Behavior, 41*, 51-61.
Malan, D. J., & Leitner, H. H. (2007). Scratch for budding computer scientists. *ACM SIGCSE Bulletin, 39*(1), 223-227.
Marimuthu, M., & Govender, P. (2018). Perceptions of Scratch Programming among secondary school students in KwaZulu-Natal, South Africa. *The African Journal of Information and Communication* (AJIC), *21*, 51-80.
Martínez-Valdés, J. A., Velázquez-Iturbide, J. Á., & Hijón-Neira, R. (2017, 18–20 October). *A relatively unsatisfactory experience of use of Scratch in CS1*, Cádiz, Spain.
Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011, 27–29 June). Habits of programming in Scratch. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* (pp. 168-172). ACM.
Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education, 23*(3), 239-264.
Mladenović, S., Krpan, D., & Mladenović, M. (2016). Using games to help novices embrace programming: from elementary to higher education. *International Journal of Engineering Education, 32*(1), 521-531.
Moreno, J., & Robles, G. (2014, 22–25 October). Automatic detection of bad programming habits in scratch: A preliminary study. *Paper presented at the Frontiers in Education Conference* (FIE), IEEE.
Olelewe, C. J., & Agomuo, E. E. (2016). Effects of B-learning and F2F learning environments on students' achievement in QBASIC programming. *Computers and Education, 103*, 76-86.
Ouahbi, I., Kaddari, F., Darhmaoui, H., Elachqar, A., & Lahmine, S. (2015). Learning Basic Programming Concepts by Creating Games with Scratch Programming Environment.

Procedia-Social and Behavioral Sciences, 191, 1479-1482.

Papert, S., & Harel, I. (1991). Situating constructionism. *Constructionism, 36*(2), 1-11. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM, 52*(11), 60-67.

Schunk, D. H. (2014). *Learning theories: An educational perspective* (6th ed.). Pearson Education Limited.

Shapiro, R. B., & Ahrens, M. (2016). Beyond blocks: Syntax and semantics. *Communications of the ACM, 59*, 39-41.

Topalli, D., & Cagiltay, N. E. (2018). Improving programming skills in engineering education through problem-based game projects with Scratch. *Computers and Education*, *120,* 64-74.

Vujošević-Janičić, M., & Tošić, D. (2008). The role of programming paradigms in the first programming courses. *The Teaching of Mathematics, 11*(2), 63-83.

Weintrop, D., & Wilensky, U. (2015, 21–24 June). To block or not to block, that is the question: students' perceptions of blocks-based programming. In *Proceedings of the 14th International Conference on Interaction Design and Children* (pp. 199-208). ACM.

Weintrop, D., Holbert, N., Wilensky, U., & Horn, M. (2012). *Redefining constructionist video games: Marrying constructionism and video game design. Paper presented at the Proceedings of the Constructionism 2012 Conference*. Athens, Greece.

Yurdugül, H., & Aşkar, P. (2013). Learning programming, Problem solving and gender: A longitudinal study. *Procedia – Social and Behavioral Sciences, 83*, 605-610. doi:10.1016/j.sbspro.2013.06.115