Research in Learning Technology
Vol. 26, 2018

ASSOCIATION FOR LEARNING TECHNOLOGY ALT

ORIGINAL RESEARCH ARTICLE

Exploring perceived cognitive load in learning programming via Scratch

Ünal Çakiroğlu[a]*, S. Sude Suiçmez[b], Yılmaz B. Kurtoğlu[c], Ayhan Sari[d], Suheda Yildiz[e] and Mücahit Öztürk[f]

[a]*Department of Computer and Instructional Technology Education, Fatih Faculty of Education, Karadeniz Technical University, Trabzon, Turkey;* [b]*Ministry of Education, Rize, Turkey;* [c]*Recep Tayyip Erdoğan University, Rize, Turkey;* [d]*Ministry of Education, Ordu, Turkey;* [e]*Ministry of Education, Rize, Turkey;* [f]*Ortaköy Vocational School, Aksaray University, Aksaray, Turkey*

(Received 8 July 2016; final version received 6 April 2018)

The purpose of this study is to investigate the perceived cognitive load and its effects on the academic performance in Scratch-based programming. The four main concepts of programming (sequences, operators, conditions and loop) were delivered in the instructional package. Participants were 12 sixth-grade students enrolled at a public secondary school. The results from quantitative and qualitative instruments indicated that students' perceived cognitive loads were close to each other among four programming concepts. The attractive interface of Scratch was somewhat useful but some parts of the interface were problematic for achieving the programming tasks. This study concludes with suggestions for Scratch practitioners and researchers to pay attention to the sources of cognitive load effects.

Keywords: cognitive load; programming course; Scratch; block based programming; primary school students

Introduction

Over the past several years, teaching programming to children has become widespread (Benton *et al*. 2017) and computational thinking has received considerable attention (Grover and Pea 2013; Barr and Stephenson 2011). Wing (2006) points out that computational thinking includes some mental tools for problem-solving related to the programming concepts. Considering its great potential in developing computational, critical and algorithmic thinking (Lee, Martin, and Apone 2014), higher-order thinking and problem-solving (Fessakis *et al*. 2013) researchers suggest starting teaching programming at early ages to young students (Govender and Grayson 2006; Jenkins 2002; Proulx 2000). However, in prior studies, some difficulties were addressed to learn programming syntax or deal with error messages in text-based programming (Lewis 2010; Resnick et al. 2009). Thus, some block-based programming environments equipped with visual tools were specifically developed for young pupils (Weintrop and Wilensky 2015). In line with this, in recent years, Scratch has become one of the most popular programming environments having over 13 million shared projects (URL 1 2016). Educators put in efforts to integrate Scratch activities into the curriculum and

*Corresponding author. Email: cakiroglu@ktu.edu.tr

Citation: Research in Learning Technology 2018, **26**: 1888 - http://dx.doi.org/10.25304/rlt.v26.1888

also teachers implement Scracth activities in schools for conceptualising programming and developing computational thinking (Fesakis and Serafeim 2009; Lewis 2010; Malan and Leitner 2007). While a large number of studies have noted the influences of using Scratch on young students' mental models of programming concepts, the conditions under cognitive processes when learning to program still remain unclear. This current study seeks cognitive processes in learning programming via Scratch to develop computational thinking skills.

Teaching programming to children with Scratch

During programming, students are exposed to computational thinking (Brennan and Resnick 2012). Computational thinking includes a set of thinking skills, habits and approaches in solving problems using computer drawing on the fundamental computer science concepts (Wing 2008). CSTA (2011) proposed a set of concepts for computational thinking, including procedures and algorithms, problem decomposition, parallelisation and synchronisation, and abstraction and data representation. Considering the idea behind computational thinking, Brennan and Resnick (2012) argued that programming with Scratch can provide opportunities for contributing to the skills related to the computational thinking.

Scratch was built on the constructionist ideas of Papert (1980) which allow learners to use the tools to work on tasks in the visual programming environment. It has become widespread interest of schools (Kalas and Benton 2017) to introduce programming concepts to students having limited programming experience (Maloney *et al*. 2010). Following Papert's ideas, researchers pointed out that suitable Logo experiences could stimulate children's cognitive development in, particularly, mathematical problem-solving. As in Logo, Scratch also serves more advanced programming environment for computational procedures and computational concepts.

Researchers suggest using Scratch for teaching programming to children because it has various kinds of tools for creating programs with the combination of graphics, animations, photos or audio (Lee 2009; Maloney *et al*. 2008). In line with this, Giannakos *et al*. (2013) reported that Scratch programming enhanced the creativity of 12-year-old-students. Resnick *et al*. (2009) suggested that Scratch provides more conceivable, meaningful and social learning environment than other platforms can. Accordingly, in an experimental study with sixth-grade students, Nam *et al*. (2010) found that using Scratch exerted a greater influence on the improvement of problem-solving skills. In addition; Benton *et al*. (2017) in their ScratchMaths project focused on learning to express mathematical ideas through programming for the young students.

Researchers argue that students should acquire procedural, conditional and analogical thinking skills in programming process (Hui Hui and Umar 2011; Law, Lee, and Yu 2010). Scratch or similar environments with visual components facilitate learning programming; however, some of the researchers feel that students still have low average scores or low problem-solving skills (Armoni, Meerbaum-Salant, and Ben-Ari 2015; Garner 2009; Kalelioğlu and Gülbahar 2014; Lister 2011). One reason for this lack of success may be the challenges in the cognitive processes during programming learning process. In this sense, cognitive load has attracted considerable attention in terms of cognitive processes of programming instruction. Since cognitive load is seen as one of the main barriers of meaningful learning, this study focused

Citation: Research in Learning Technology 2018, 26: 1888 - http://dx.doi.org/10.25304/rlt.v26.1888

on the influences of Scratch environment on cognitive processes and dealt with the theoretical aspects of cognitive load.

Cognitive load in programming

Cognitive load is defined as a kind of pressure exerted on learners' cognitive system (Leahy and Sweller 2011; Paas and Sweller 2012). This pressure is explained in the cognitive processing within the following assumptions: two channels of human information-processing system having limited capacity and limited amount of cognitive processing taking place at any time (Sweller 2010). The theory defines three different forms of cognitive load; intrinsic, extraneous and germane load. When the elements of information interact, namely, the information is relatively complex, the elements must be processed simultaneously in working memory and it may result in high intrinsic cognitive load (Sweller 2016). Extraneous cognitive load is imposed by the form and means through the material experienced (Pollock, Chandler, and Sweller 2002; Renkl and Atkinson 2003; Sweller 2010) and weak problem-solving methods in information sources to complete a learning task (van Merriënboer and Ayres 2005).

Germane cognitive load is devoted to the processing, construction and automation of schemas (Sweller 2010). For meaningful learning, researchers suggest reducing extraneous load and increasing the germane load (Paas *et al*. 2003). In the light of the assumptions, Mayer and Moreno (2003) proposed instructional principles for designing the multimedia for educational purposes. In this sense, many features of learning environments are considered to influence cognitive processing. Hence, appropriate use of colour, orientation, curvature, size, motion, positioning, shape, signals, information modes or other features may induce a lower load on working memory (Moons and De Backer 2013; Wolfe 2000).

Researchers advocated that it is difficult to reduce cognitive load during the learning process of programming (Mead *et al*. 2006; Renkl and Atkinson 2003; Stachel *et al*. 2013). Mason, Cooper, and Wilks (2015) documented that some of the programming environments may be complex and cause an adverse impact on learners' focus of attention and overloading cognitive resources for learning. In this sense, a basic idea about designing easy-to-use platforms may be taken into consideration (Oviatt 2006). Educators generally choose appropriate programing environments to help novices; however, it is unclear whether they pay attention to the cognitive effects of their interfaces. Although some marked advances have been made in programming environments for children and in the teaching methods, more research is needed in order to understand the cognitive challenges in learning programming better. Thus, this study hypothesised that cognitive load in learning programming may be caused by Scratch itself and exploring cognitive load may provide suggestions regarding programming interfaces for teachers.

Purpose of the study

This study attempts to find out how the effects of cognitive load function in the instructional process of four main concepts of programming via Scratch. The following sub-problems were guided to the study.

What kind of influences in terms of cognitive load did the students perceive?

What is the relationship between perceived cognitive load and academic performances?

Method

This exploratory case study was carried out at a secondary school in information technologies and software course during four lesson periods in 4 weeks. The course introduces basic components of information technologies and improves computational thinking skills via block-based programming environments.

Participants

A total of 12 sixth-grade students (2 males, 10 females, average age: 11–12) enrolled at a public secondary school participated in this research. Students were able to use basic office programs and they had presented only a few sample codes in the previous year; however, they had no experiences in how to code with Scratch.

Process

In a wide range of Scratch projects, it is seen that the frequently used programming concepts are sequences, loops, parallelism, events, conditionals, operators and data (Brennan and Resnick 2012). In line with this, most of the studies about learning programming focus on the basic concepts of programming such as variables, loops, conditions and controls, message passing or concurrency (Kalas and Benton 2017; Meerbaum-Salant, Armoni, and Ben-Ari 2013). In parallel with prior studies and the objectives of the course, we focused on four main concepts of introductory programming. Since sequences are key concepts in programming including a series of instructions and actions that can be executed by the computer, it was one of the main concepts of this study. Conditions, operators and loops are also taken into consideration as they are common concepts of typical programming instructions.

In the first week, the teacher introduced how to use the basic components of Scratch. Then she presented some simple examples about the four concepts. Because the present study focuses on the external cognitive load induced from Scratch environment, the tasks in the study were not so easy. The cognitive load theory suggests that the capacity for total cognitive load within working memory is limited. Hence, when intrinsic load is high, extraneous cognitive load must be lowered; when low intrinsic load occurs, high extraneous cognitive load may not hinder learning (van Merriënboer and Ayres 2005). Thus, we developed somewhat complex tasks for beginners in order to observe the influence of cognitive load induced from the programming environment itself. A brief summary of the instructional process is outlined in Table 1.

Students were asked to fill the cognitive load scale to measure their perceived cognitive load at the end of these four applications.

Instrumentation and analysis

Qualitative and quantitative data were collected and interpreted together in the study. Interviews, cognitive load scale, rubric and screenshots were used as data collecting tools.

Citation: Research in Learning Technology 2018, 26: 1888 - http://dx.doi.org/10.25304/rlt.v26.1888

Table 1. A brief summary of the instructional process.

Topic	Scratch components expected to be used	Summary of application conducted	Period
Introducing basic components of Scratch		Providing information about how to use Scratch tools in solving programming via Scratch.	First week
Sequences	Control blocks Motion blocks	In this application, students were asked to use Scratch menus at basic level for 2 h. A flying sprite was created and the students were asked to change the coordinates of the sprite for a given sprite. After practicing with the scenario, students were given an example and asked to develop the same algorithmic process individually. In this study, they were asked to jump a ball to proper coordinates described in a scene.	Second week
Operators	Control blocks Look blocks Variables blocks Numbers blocks	In this application, the teacher presented a sample activity including adding and subtracting. In the second hour, a calculator application in which the multiplication and division processes were added into the scenario was discussed with students.	Second week
Conditions (If)	Control blocks Look blocks Motion blocks Sensing blocks	Examples 'condition' were presented to the students at the beginning of the lesson. An object was created and a movement feature was assigned to the object with the arrow keys. The object was returned to the starting point when a different object touched it. In the second hour, students were shown a maze game application. Then, they were asked to develop a similar game individually.	Third week
Loop	Control blocks Look blocks Sensing blocks Variables blocks	In this application, basic loop activities were discussed in the first lesson using Scratch. A number of objects were created on the scenes which are continuously moving with the arrow keys. In the second hour, students were shown a shark game application including a scenario in which a small fish is trying to catch the shark. Scenario also has a game scoring. In order to complete this activity, loop procedures and control blocks were required.	Fourth week

Cognitive load scale

Cognitive load scale was provided by Paas and Van Merrienboer (1993) and was adapted to Turkish by Kılıç and Karadeniz (2005). The scale includes one item (scored from 1 to 9) about the students' efforts to achieve the tasks. The scale was used to measure perceived cognitive load in similar studies (Renkl and Atkinson 2003). Under the *loop* concept, the item in the cognitive load scale becomes 'How much effort did you spend for this work?'. Students filled the scale at the end of the four lessons. After the instructional process, the average of measured scores in four lessons was considered as cognitive load scores (CLS).

Rubrics

In order to assess the students' solutions for the programming problems, we developed rubrics for each concept (sequences, conditions, operators and loops) by taking the views of three field experts at programming and Scratch. The rubric scores of students were calculated for each concept and used as academic achievement scores (AAS).

Screenshots

Students' navigations in the Scratch activities were recorded using a screenshot recorder. In this way, we collected useful data on the components and navigations while students were solving problems in Scratch. Students' efforts on the tasks were analysed by searching answers for the following questions. How much time did you spend on sub-task? Which components did you use and how did you use them in order to complete the sub-task? What were the challenges using Scratch during completing the sub-task?

Interviews

Semi-structured interview questions were developed with the help of an expert and used to elaborate the data from the scale and the rubrics. The questions were developed within the framework of four main topics and related to how students used Scratch components in the tasks.

Results

This section is organised in relation to the two research questions: (1) Students' perceived cognitive loads and (2) the relations between the measures of cognitive load and academic achievements.

Perceived cognitive load on using Scratch components

In order to explain the students' actions on the tasks (to find answers to why and how they perceived cognitive load), we determined the frequency of the blocks and sprites used in the tasks as well as the flows of the sprite on the blocks. The mean values of the participants' perceived CLSs were calculated and assigned as CLSs. Similar

Citation: Research in Learning Technology 2018, 26: 1888 - http://dx.doi.org/10.25304/rlt.v26.1888

Table 2. Students' actions in using the Scratch components.

Topic	Sequences	Operators	Condition (If)	Loop
CLS	5.08	4.58	4.08	3.33
Blocks used	Control, Motion Looks	Control, Sensing Variables, Operators	Control, Motion Sensing, Looks	Control, Looks Motion, Sensing Variables
Sprite used in the codes	*Sprite 1 Stage*	*Sprites 1, 2, 3, 4, 5*	*Sprite 1*	*Sprites 1, 2*
Steps used in blocks in which Sprites were used	*Sprite 1* is used in the sequence of Control and Motion, then used in Stage; Control, and Looks	*Sprite 1* is used in the sequence of Control, Sensing and Variables and *Sprite 2, 3, 4, 5* are used in Control, Variables, Operators and Variables	*Sprite 1* is used in Control, Motion, Sensing and Looks	*Sprite 1* is used in Control, Looks and Motion, then *Sprite 2* is used in Control, Motion, Sensing, Variables and Looks

analysis on CLSs is provided in some other studies (İzmirli and Kurt 2016). We examined the navigations followed by students in the tasks from screenshots and we used the interviews to explain these navigations. The screenshots and students' comments on the interviews were interpreted together. The CLSs for all concepts are shown in Table 2.

Navigations on the tasks

Students' actions were noticed as navigations among the blocks and briefly summarised in four concepts in Table 2.

Table 2 indicates that the CLSs of the concepts *sequences* and *operators* are quite higher than the average CLSs of *conditions (if)* and *loops*. In the *sequences* concept, Sprite 1 is used in the Control, Motion and Looks blocks, and Sprite 2 is used in the Control, Motion, Sensing and Looks in Condition blocks. It is seen that in the *sequences* concept, students used three blocks (Control, Motion, Looks) and two Sprites. In the *operators* concept, five different sprites were used in the various components of Control, Sensing, Variables and Operators blocks. In the *loop* concept, students needed to use five blocks (Control, Looks, Motion, Sensing, Variables) and two Sprites. Although the tasks in the loop were more complex than the tasks of other concepts, the average CLSs were found low in the *loop* concept. At this point, the interview data revealed that after a short adaptation period, students could use Scratch components easily. Since they were accustomed to the interface from previous phases, they could work on the next tasks by adding new sprites. In this sense, S6 stated that 'It was difficult to solve the problem in the first activity but I understood the sequences better in the next week. In the first week, I assigned wrong coordinates to code blocks by moving the mouse on the screen unconsciously; I corrected it in time'. In addition, S5 specified that 'I had difficulty using the program in the first week, but I could easily find the codes and put them in order when I practiced in the second week, I could do

it better after I used in the first task'. Whereas the students easily completed the task related to the conditions (if), they had difficulties especially in the tasks requiring intricate codes. S9 expressed the cognitive load which he confronted in if blocks as follows 'Coding long programs were challenging. It was easy to write the codes with single lines, but it was difficult to develop multi-conditional if codes'.

In the four concepts, we have observed that students provided more effort in the *operators* than the *condition (if)* concept. They encountered with difficulties, especially in creating variables and using operators on the Scratch blocks. Even though students worked on the operators only with basic addition operations, they took a long journey to complete the tasks. Figures 1, 2, 3 and 4 present the blocks, sprites and students' navigations in the tasks. The below example includes the solution steps for the activity of creating a calculator about the *operators* concept.

Firstly, students created number 1, number 2 and the result variable for the operators activity by utilising variables code block.

After defining the variables, students wrote the code by clicking the code part on the control block. Later, in order to enter a number, students assigned the sprite to give the command 'enter the number'. Thus, they used the code part 'ask and wait'

Figure 1. Defining variables.

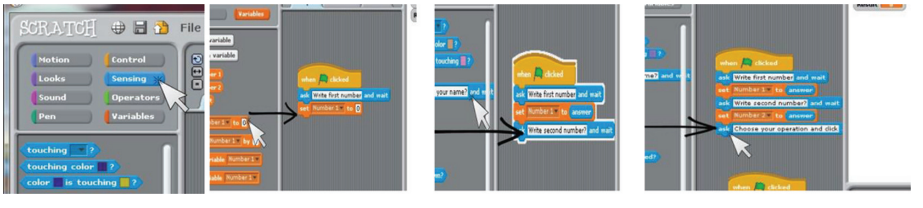

Figure 2. Students' actions when clicked on the buttons.

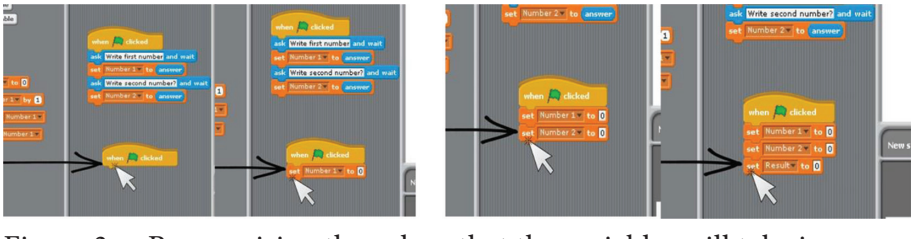

Figure 3. Reorganising the values that the variables will take in process.

Citation: Research in Learning Technology 2018, 26: 1888 - http://dx.doi.org/10.25304/rlt.v26.1888

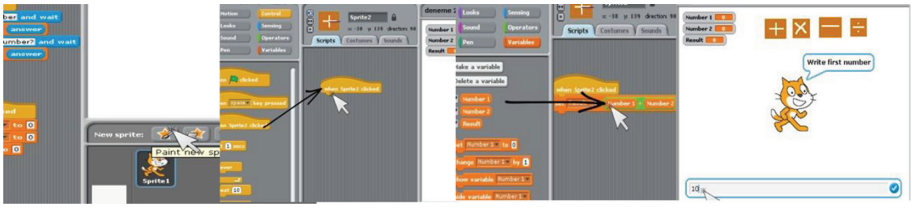

Figure 4. The last step for coding the calculator.

on the sensing block and wrote the expression of 'enter the number' inside. Then, they clicked the code part 'do ... instead of the variable ...' on the variable block and located on the code part 'answer' inside the sensing block. At the last step, they used the code part 'ask and wait' on the sensing block to make sprite wait for the answer about the operation.

Then, students inserted a code to reset the initial values of the variables. To achieve it, under the code 'when clicked' on the 'control' blocks, students used code 'reset for each variable' on the 'variables' code block.

Students developed new sprites for the four basic operations: addition, subtraction, multiplication and division. Then, they wrote codes by entering in the code area of each sprite. For instance, first they followed the code part 'when clicked on the sprite' on the *control* code block and then, they utilised the code part 'do the result variable ...' from the *variables* code block. After this stage, they assigned numbers to the variables as 'number 1' and 'number 2' in the *variables* code block. These steps were repeated for the other three operations. The appropriate operation (add, subtract, multiply or divide) was used for each operation on the *operators* code block and then, they ran the program. After the message 'Enter the first number', the user entered the value for the variable which was assigned as the first and also the second variable. Later, when the user clicked on the addition, the sum of the two numbers entered appeared as the result of operation.

Furthermore, we analysed the screenshots to explain the navigations followed by the students to create the sprites. Surprisingly, sometimes some basic processes require complex navigations. For instance, the screenshots taken from the operator section of Scratch indicated that in order to achieve a basic addition operation, a logical order of control, variable, operators and sensing blocks are required. Although the operation of '10 + 20' is an easy addition operation that almost all students can answer in seconds, the flow of the blocks and the necessary code for the operation seems to be complicated to the students.

By analysing the navigations in the screenshots, we defined the ways which were frequently followed to provide codes for the tasks. The steps are outlined in Figure 5.

The average length of the codes for the concepts was L_Condition > L_Loop > L_Operations > L_ Sequences. The length of the codes is not directly related to CLSs. While the average CLS of the *sequences* concept has the highest CLS value, the average length of codes about *sequences* is the shortest. Also, while the codes in *condition (if)* were somewhat long, the CLSs was not so high.

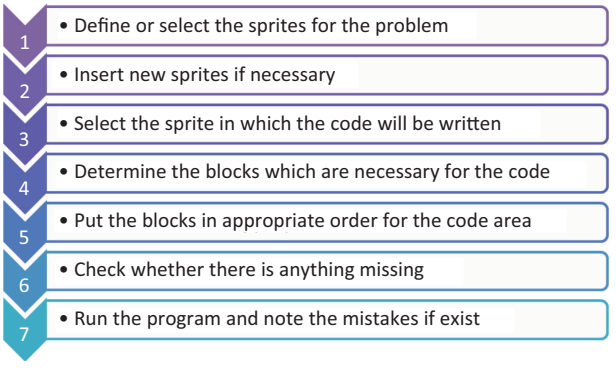

Figure 5. The steps followed by students to achieve the tasks.

Table 3. Achievement scores.

Topic	Item	Sub-level	Average achievement score	Maximum total score
Sequences	1	Selecting sprite and scene	9.66	10
	2	Moving the sprite in the appropriate period	30	40
	3	Shifting to the correct phase	17.91	25
	4	Providing codes in a hierarchical order	23.75	25
Operators	1	Defining the variables	15	15
	2	Responding to variable assignment	15.83	20
	3	Defining arithmetic operations	25.41	40
	4	Providing codes in a hierarchical order	19.16	25
Conditions (if)	1	Assigning motion task to the keys	19.58	20
	2	Adding barriers to the charter command	28.75	30
	3	Adding target to the charter command	27.5	30
	4	Providing codes in a hierarchical order	17.91	20
Loop	1	Moving the big and small fish in the right direction	24.16	30
	2	Defining the score variable and creating the correct code	20	20
	3	Creating the repeating command	23.75	25
	4	Providing codes in a hierarchical order	20.41	25

Perceived cognitive load on the tasks

Students' perceived CLSs were calculated through the cognitive load scale and the achievement scores were determined via the rubrics. The evaluation criteria for the four main topics and the achievement scores in the rubric are presented in Table 3.

Citation: Research in Learning Technology 2018, 26: 1888 - http://dx.doi.org/10.25304/rlt.v26.1888

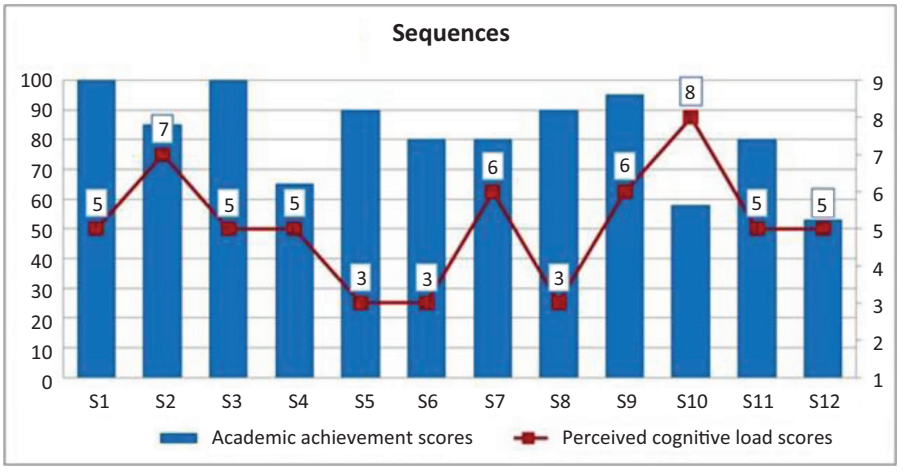

Figure 6. Relations between AAS and CLS (Sequences).

Only two students got 8 points, and ten students got 10 points in the first item of the *sequences* concept. The average score for 'Moving the sprite in the appropriate period' was 30 and the average score for shifting to the correct phase was 17.91 out of 25 points. In this sub-level, three students got 25 points and only two students got 0 points. In the *operators* concept, all of the students got full points in 'Defining the variables'. 'Variable assignment' was evaluated out of 20 points in which five of the students got more than 16 points and seven students had less than 16 points. In this sub-level, five students had full score and seven students got less than 20 points. In the evaluation of *conditions (if)* topic, the highest average scores was 28.75 out of 30 points on the 'Adding barriers to the charter command'. Ten students got full points in this sub-level. Moreover, in the *loop* topic, all of the students could correctly complete the sub-level of 'Defining the score variable and creating the correct code' which was evaluated out of 20 points.

Relationship between perceived cognitive load and academic performances

CLSs ranged from 1 to 9, the achievement scores ranged from 0 to 100. Therefore, in order to provide a better understanding of the relations, we have made a comparison of these two different measures mapping them in the same figures.

Sequences

In the *sequences* concept, the task was to create a flying sprite. Students were asked to use several motion blocks and change the background at the end of the movement. The correlations of AAS-CLS during the tasks are shown in Figure 6.

According to the measurements, AAS = 81.33 and the average CLS = 5.08. The students who had low perceived CLS generally exhibited a higher score of AAS and vice versa. For instance, whereas S1 and S3 had a CLS of 5, their AAS was 100. In contrast, when the CLS increased, AAS generally decreased in the *sequences* concept; (S2: CLS = 7, AAS = 85) and (S10: CLS = 8, AAS = 58).

Operators

In the *operators* concept, students were asked to use Scratch operators to create a simple calculator. In order to complete the task, they were required to create and use two number variables and one resulting variable. They were asked to do some arithmetical operations with the variables, store the values of the variable and also use appropriate code blocks in order to achieve the task. Figure 7 outlines the relationship between AAS and CLS in this concept.

In the *operators* concept, the AAS was 75.41 and the CLS was 4.58. No relationship between AAS and CLS was found in Figure 7. For example, despite having similar CLS, S4 had 45 points and S10 had 60 points of achievement score. Furthermore, S4 and S7 have got the same CLSs; however, their AASs are explicitly different (45 and 100, respectively).

Condition (if)

Relationship between AAS and CLS of the *condition* concept is illustrated in Figure 8.

The measurements in the *condition (if)* concept indicated that the AAS of the group was quite high, which was 93.75, and the CLS of the group was relatively low, which was 4.08. According to these measures, in the *condition* concept, students with high AAS generally got low CLS. Where only six students' achievement scores were 100 points, most of them had low CLS. (S2, S3, S4 and S12 had the lowest CLS value in the score range of 2–3). It is remarkable that one of the students got the highest CLS (7) and the lowest AAS (68).

Loop

The measures about *loop* concept are shown in Figure 9. The relationship between AAS and average CLS was slightly surprising.

According to the measures in *loop* concept, the AAS was 89.2 points and the CLS was 3.33. Interestingly, the average value of CLS in this concept was lower when

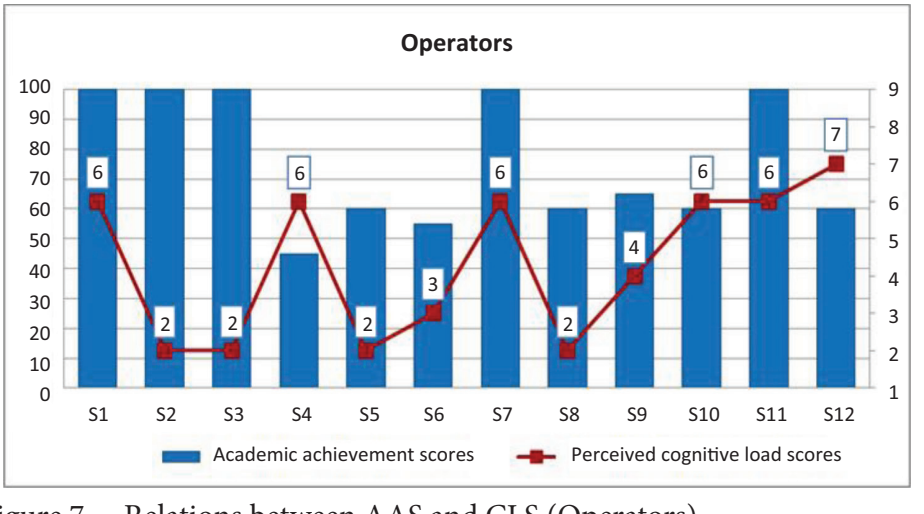

Figure 7. Relations between AAS and CLS (Operators).

Citation: Research in Learning Technology 2018, 26: 1888 - http://dx.doi.org/10.25304/rlt.v26.1888

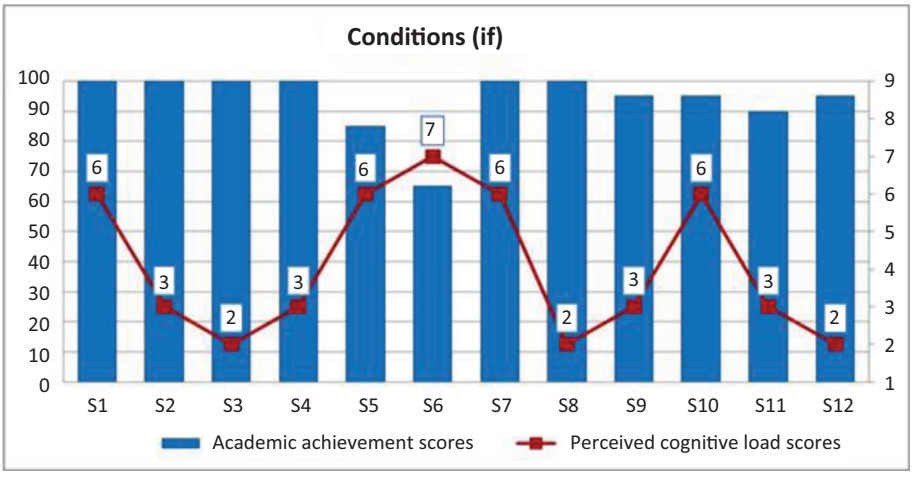

Figure 8. Relations between AAS and CLS (Conditions-If).

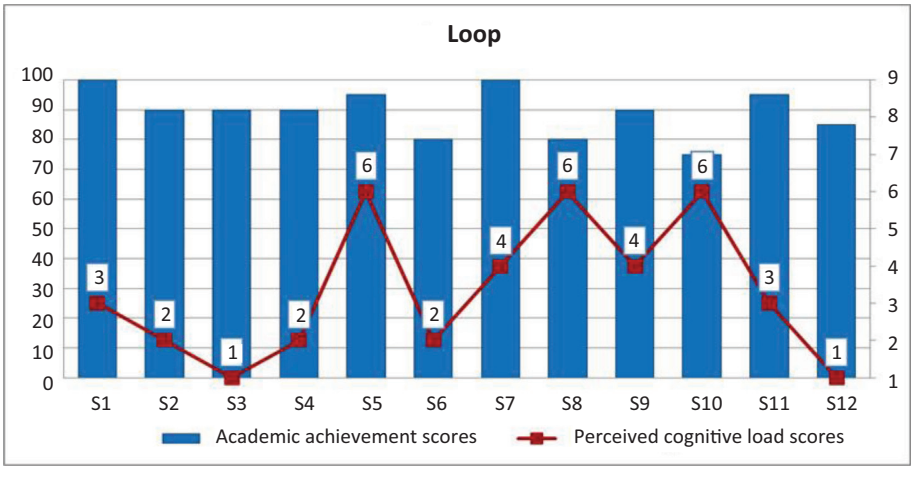

Figure 9. Relations between AAS and CLS (Loop).

compared to the other concepts. On the contrary, AAS in the *loop* concept was found to be quite high and the lowest AAS was 75 points. Most of the students got 90 points as achievement score. Surprisingly, S5 got a relatively higher score of cognitive load (6) than other students and his AAS was also higher than the others' scores. The scores of S8 and S10 support the idea that when CLS increases, AAS decreases. For example, as two students (S1 and S7) got the full score of academic achievement (100 points), S1 got the average CLS of 3 and the other, S7, got the average CLS of 4 and their average CLSs were not the lowest. As a result, it is not easy to define an accurate relationship between the CLS and AAS in this concept. Including all the concepts, the relationship between AASs and CLSs is illustrated in Figure 10.

It is found that except the *loop* concept, there is an inverse relationship between average AASs and CLSs. The lowest CLS was induced in the *loop* concept, but the

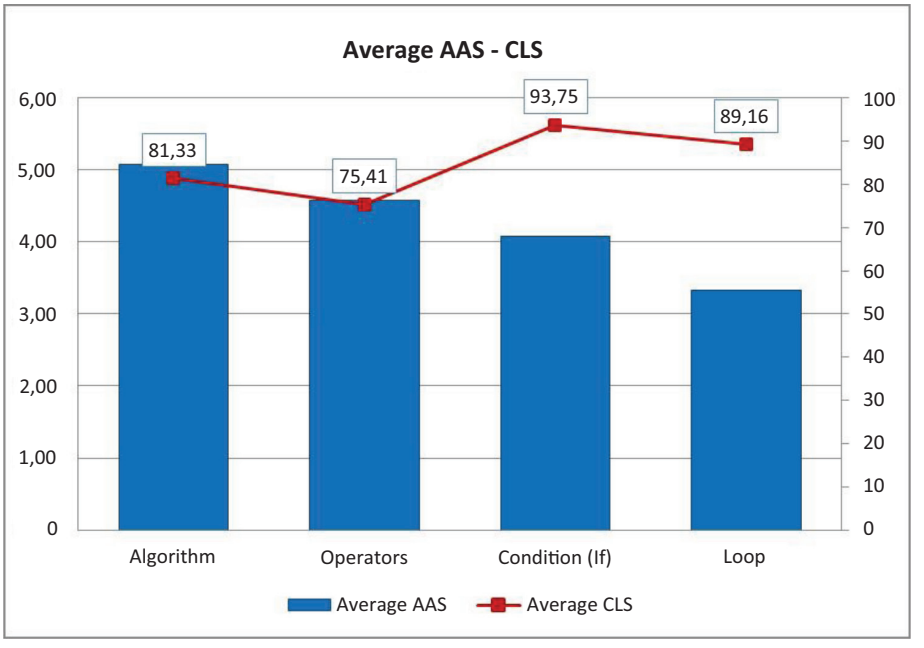

Figure 10. Average achievement and CLS in the four concepts.

highest average AAS was observed in the *sequences* concept. The AAS from highest to lowest scores was for *condition (if)*, *loop*, *sequences* and *operators* concepts, respectively.

In other words, it can be concluded that the CLS decreases when the students become more experienced in using Scratch components. In a weekly review of the findings, the highest CLS was observed in the third week in the *condition (if)* and the lowest CLS was observed in the second week in the *operators* concept. In this case, the findings from the interviews revealed that the following concepts were covered after *sequences* concept and at that time students had already become more familiar with using Scratch components.

Discussion

Many young students who have no experience may hit a cognitive barrier at the beginning of programming learning process (Smith, Cypher, and Tesler 2000). During programming, students generally want to achieve the tasks or solve problems as quickly as possible. In this sense, Scratch as a visual programming platform may be used in order to visualise the programming process. It may be useful to visualise not only the typical programming components (variables, arrays, I/O components, etc.) but also the main programming concepts (sequences, operators, conditions, loops, etc.). Resnick *et al*. (2009) stated that Scratch blocks may facilitate the programming mechanics by eliminating syntax errors, providing feedback on the location of command blocks and giving immediate feedback to the appropriate codes. In addition, Gadanidis *et al*. (2017) reported that they also affords new approaches to mathematics

Citation: Research in Learning Technology 2018, 26: 1888 - http://dx.doi.org/10.25304/rlt.v26.1888

problem-solving. In this study, using Scratch components somehow could shorten the necessary codes required to achieve the tasks. In a similar vein, Meerbaum-Salant, Armoni, and Ben-Ari (2013) argued that Scratch may eliminate the extraneous cognitive load induced from syntax and facilitate the programming process by the visualisation. However, the findings of this study indicated that it is not easy to achieve the tasks based on the logical and conceptual knowledge of programming using Scratch.

In this study, students found it difficult to find and use appropriate components of Scratch and to provide a solution strategy for the problem at the same time. Thus, this was one reason for them perceiving cognitive load. Especially in the *sequences* concept, students found it difficult to remember which code was under which block. Then, in the following tasks, students were asked to put the codes in the blocks in order to create new code blocks. Most of them could create separate code blocks such as 'clicked' and 'pressed the arrow keys' in which some extra effort was required to find the necessary block and join it with others. It was difficult for the students to remember the sequence of the components to decide which tool was appropriate for the task. For example, a simple sequence of Scratch instructions may appear, in which one block initialises the position of a sprite, another block, its direction, and a third, its costume. In the study, when a number of commands (using *loop* in the *conditions* block or using *conditions* in *condition* block) were asked to use in the code, some students were confused which blocks they needed to use for the solution. In doing so, some students provided extra efforts for using the code block itself and also covered it with master code blocks.

Another extraneous cognitive load was induced from allocating the scene and sprites on the Scratch interface, so some sprites were sometimes placed out of the scene. In addition, in this study colours, buttons and drag/drop options were perceived as extraneous cognitive load sources. In this regard, Moons and De Backer (2013) identified some extraneous cognitive load caused by the programming editor interfaces such as colour (hue, lightness and colourfulness), orientation, size, motion, relative positioning and shape of the components.

This study addressed some evidences that cognitive load may occur in various concepts of basic programming process. In this study, in the *sequences* concept in which students created a flying sprite, the average CLS and total achievement score had an inverse relationship with each other. In the *sequences* concept, students with low scores of achievement generally provided more effort than others. In addition, average CLSs in the *condition* concept were relatively low because only control and motion blocks were used in this concept. This is because various sub-scripts that are under the motion block may increase cognitive load. In addition, the sequence of the codes also includes more complex codes than other concepts; therefore, it may be difficult for students to imagine the location of the code blocks in the *condition* concept. Average CLSs in the *loop* concept were also quite low. Hence, the results of this study suggest that except the *loop* concept, students' perceived CLSs have considerably an inverse relationship with their academic achievements. Creating various codes with different sprites for the first time might have negatively affected the students' academic achievement. Overall, it was observed that the perceived CLSs decreased gradually during the implementation. At this point, the nature of the concept and the given problem about the concept affected the level of perceived cognitive load. A particular phenomenon identified by Sweller (2010) can also explain this effect in which cognitive load can decrease as the learner becomes more expert and familiar with the particular environment for learning. Sweller (2016) explains such effect as worked examples effect; that

is, those who study with worked examples perform better on problems than learners who solve the same problems themselves. A conclusion may be derived from this study that the expertise on the use of environment and the relation of tools, components and tasks may also facilitate solving problems in Scratch. Sweller (2016) argued that this increased expertise reduces the element interactivity and only a few memory resources become sufficient during problem-solving.

The duration of the period in which students' learning was evaluated is limited. However, the time period was enough to accustom to use the Scratch tools and environment in this study. One reason of the cognitive load induced in the following weeks by some students may be the changing nature of the programming concept. Although some experiences about the difficulties of the concepts (operator, condition, loop and combinations of these) are noticed, some evidence is still required to define the possible load inherited from the concepts. Thus, the results may be helpful for cognitive load-programming studies. Another reason is split attention during solving problems in Scratch. Sweller (2016) exemplifies this as split attention between the statements and diagrams or between different categories of statements in geometry or physics. Similar appearance emerges in Scratch in which disparate sources of information/tools should be mentally integrated to reduce extraneous cognitive load.

In addition, independently from context, students who were accustomed to the interface perceived less cognitive effort. In a recent study, Sweller (2016) argues this effect as the expertise reversal effect. It is about the perceptions of element interactivity. When students are novices, element interactivity can be considered high and when they become experts, element interactivity is likely to be low. In this study, the reason for their lower AASs may be due to the fact that various code blocks were being used in the *operators* concept and assigning multiple variables and using variables in the process may have been challenging for the students.

In fact, this research is exploratory in nature and is limited for generalisation, but it has some opportunities for future search. Since the sample size was small (n = 12) and concepts of programming were limited, a larger sample size would increase the sensitivity of the analysis. Also, changing the complexity of the tasks in the expanded programming topics may suggest new insights. Although the results in four the concepts may not reflect the whole programming process, various data collecting tools in the study provided considerable evidence on the cognitive process of the children in learning programming.

Conclusions and recommendations

A growing number of K-12 schools have begun to use Scratch as a first step platform for teaching programming. This study suggests that although it presents many advantages for understanding basic programming concepts and problem-solving, Scratch also has some unusable components which may lead to cognitive load.

The highest cognitive load level was recorded in the *sequences* concept and the lowest perceived cognitive load was observed in the *loop* concept. Perceived CLSs were generally inversely related to the academic performances, however, not regularly. When students were accustomed to use the Scratch components, their perceived CLSs gradually decreased.

In conclusion, the results revealed that the interface of Scratch is attractive and valuable, but it is not easy to use the interface for the tasks constructed with nested

Citation: Research in Learning Technology 2018, 26: 1888 - http://dx.doi.org/10.25304/rlt.v26.1888

concepts of programming. In this sense, this study has some implications about teaching programming through Scratch. First of all, teachers should pay attention to students' prior experiences when they use Scratch for the first time. After introducing the interface, teachers can continue with the tasks in a sequence from easy to complex. Developing simple tasks for students may facilitate the programming process by reducing the necessary effort for the tasks. For instance, problems which require using too many motion blocks may cause high cognitive load, so tasks should be planned including motion blocks in a balanced way. To sum up, the results of this study support the idea that problem-solving via programming is something more than the visualisation of the problems. Thus, we hope this study may contribute to the efforts on creating learning environments for programming with Scratch for children.

References

Armoni, M., Meerbaum-Salant, O. & Ben-Ari, M. (2015) 'From scratch to "real" programming', *ACM Transactions on Computing Education (TOCE*), vol. 14, no. 4, pp. 25–33.

Barr, V. & Stephenson, C. (2011) 'Bringing computational thinking to K-12: what is involved and what is the role of the computer science education community?', *Acm Inroads*, vol. 2, no. 1, pp. 48–54.

Benton, L., *et al*., (2017) 'Bridging primary programming and mathematics: some findings of design research in England', *Digital Experiences in Mathematics Education*, vol. 3, no. 2, pp. 115–138.

Brennan, K. & Resnick, M. (2012) 'New frameworks for studying and assessing the development of computational thinking', *Proceedings of the 2012 Annual Meeting of the American Educational Research Association*, Vancouver, Canada, pp. 1–25.

CSTA. (2011) 'Computational thinking in K–12 education leadership toolkit', [online] Available at: http://csta.acm.org/Curriculum/sub/CurrFiles/471.11CTLeadershiptToolkit-SP-vF.pdf

Fesakis, G. & Serafeim, K. (2009) 'Influence of the familiarization with scratch on future teachers' opinions and attitudes about programming and ICT in education', *ACM SIGCSE Bulletin*, vol. 41, no. 3, pp. 258–262.

Fessakis, G., Gouli, E. & Mavroudi, E. (2013) 'Problem solving by 5–6 years old kindergarten children in a computer programming environment: a case study', *Computers & Education*, vol. 63, no. 4, pp. 87–97.

Gadanidis, G., *et al*., (2017) 'Computational thinking in mathematics teacher education', *Contemporary Issues in Technology and Teacher Education*, vol. 17, no. 4, pp. 458–477.

Garner, S. (2009) 'Learning to program from Scratch', *Ninth IEEE International Conference on Advanced Learning Technologies*, IEEE, Riga, Latvia, pp. 451–452.

Giannakos, M., Hubwieser, P. & Chrisochoides, N. (2013) 'How students estimate the effects of ICT and programming courses', *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, ACM, Denver, Colorado, USA, pp. 717–722.

Govender, I. & Grayson, D. (2006) 'Learning to program and learning to teach programming: a closer look', in *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2006*, eds E. Pearson & P. Bohman, AACE, Chesapeake, VA, pp. 1687–1693.

Grover, S. & Pea, R. (2013). 'Computational thinking in K–12: A review of the state of the field'. *Educational Researcher*, vol. 42, no. 1, pp. 38–43.

Hui Hui, T. & Umar, I. N. (2011) 'Does a combination of metaphor and pairing activity help programming performance of students with different self-regulated learning level?', *The Turkish Online Journal of Educational Technology*, vol. 10 no. 4, pp. 121–129.

Izmirli, S. & Kurt, A. A. (2016) 'Effects of modality and pace on achievement, mental effort, and positive affect in multimedia learning environments', *Journal of Educational Computing Research*, vol. 54, no. 3, pp. 299–325.

Jenkins, T. (2002) 'On the difficulty of learning to program', *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, pp. 53–58, Loughborough, UK.

Kalas, I. & Benton, L. (2017) 'Defining procedures in early computing education', *IFIP World Conference on Computers in Education*, Springer, Cham, pp. 567–578.

Kalelioğlu, F. & Gülbahar, Y. (2014) 'The effect of teaching programming via scratch on problem solving skills: a discussion from learners' perspective', *Informatics in Education*, vol. 13, no. 1, pp. 33–50.

Kılıç, E. & Karadeniz, S. (2005) 'Specifying students' cognitive load and disorientation level in hypermedia', *Educational Administration: Theory and Practice*, vol. 40, no. 1, pp. 562–579.

Law, M. Y., Lee, C. S. & Yu, Y. T. (2010) 'Learning motivation in e-learning facilitated computer programming courses', *Computers & Education*, vol. 55, no. 1, pp. 218–228.

Leahy, W. & Sweller, J. (2011) 'Cognitive load theory, modality of presentation and the transient information effect', *Applied Cognitive Psychology*, vol. 25, pp. 943–951.

Lee, I., Martin, F. & Apone, K. (2014) 'Integrating computational thinking across the K-8 curriculum', *ACM Inroads*, vol. 4, pp. 64–71.

Lee, J., Jr. (2009). *Scratch Programming for Teens*. eds J. Davidson, Cengage Learning, Boston, pp. 27–44.

Lewis, C. M. (2010) 'How programming environment shapes perception, learning and goals: logo vs. Scratch', *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, ACM, New York, USA, pp. 346–350.

Lister, R. (2011) 'Computing education research programming, syntax and cognitive load', *ACM Inroads*, vol. 2, no. 2, pp. 21–22.

Malan, D. J. & Leitner, H. H. (2007) 'Scratch for budding computer scientists', *ACM SIGCSE Bulletin*, vol. 39, pp. 223–227.

Maloney, J., *et al*., (2008) 'Programming by Choice: urban youth learning programming with Scratch', *ACM SIGCSE Bulletin*, vol. 40, no. 1, pp. 367–371.

Maloney, J., *et al*., (2010) 'The scratch programming language and environment', *ACM Transactions on Computing Education (TOCE*), vol. 10, no. 4, p. 16.

Mason, R., Cooper, G. & Wilks, B. (2015) 'Using Cognitive Load Theory to select an environment for teaching mobile apps development', in D. D'Souza & K. Falkner (eds), *Proceedings of the 17th Australasian Computing Education Conference*, Sydney, Australia, 27–30 January, The Conference in Research and Practice in Information Technology (CRPIT) series; 160, Australian Computer Society, Sydney, Australia, pp. 47–56.

Mayer, R. E., & Moreno, R. (2003). 'Nine ways to reduce cognitive load in multimedia learning'. *Educational Psychologist*, 38(1), 43–52.

Mead, J., *et al*., (2006) 'A cognitive approach to identifying measurable milestones for programming skill acquisition', *ACM SIGCSE Bulletin*, vol. 38, no. 4, pp. 182–194.

Meerbaum-Salant, O., Armoni, M. & Ben-Ari, M. (2013) 'Learning computer science concepts with Scratch', *Computer Science Education*, vol. 23, no. 3, pp. 239–264.

Moons, J. & De Backer, C. (2013) 'The design and pilot evaluation of an interactive learning environment for introductory programming influenced by cognitive load theory and constructivism', *Computers & Education*, vol. 60, no. 1, pp. 368–384.

Nam, D., Kim, Y. & Lee, T. (2010) 'The effects of scaffolding-based courseware for the Scratch programming learning on student problem solving skill', in *Proceedings of the 18th International Conference on Computers in Education*, eds S. L. Wong *et al*., Asia-Pacific Society for Computers in Education, Putrajaya, Malaysia, pp. 723–727.

Oviatt, S. (2006) 'Human-centered design meets cognitive load theory: designing interfaces that help people think', in *Proceedings of the 14th ACM International Conference on Multimedia*, ACM, Santa Barbara, CA, USA, pp. 871–880.

Citation: Research in Learning Technology 2018, 26: 1888 - http://dx.doi.org/10.25304/rlt.v26.1888

Paas, F., Renkl, A. & Sweller, J. (2003) 'Cognitive load theory and instructional design: recent developments', *Educational Psychologist*, vol. 38, no. 1, pp. 1–4.

Paas, F. & Sweller, J. (2012) 'An evolutionary upgrade of cognitive load theory: using the human motor system and collaboration to support the learning of complex cognitive tasks', *Educational Psychology Review*, vol. 24, pp. 27–45.

Paas, F. G. W. C. & Van Merrienboer, J. J. G. (1993) 'The efficiency of instructional conditions: an approach to combine mental effort and performance measures', *Human Factors*, vol. 35 no. 4, pp. 737–743.

Papert, S. (1980) *Mindstorms: Children, Computers, and Powerful Ideas*, Basic Books, Inc., New York,USA

Pollock, E., Chandler, P. & Sweller, J. (2002) 'Assimilating complex information', *Learning and Instruction*, vol. 12, pp. 61–86.

Proulx, V. K. (2000). 'Programming patterns and design patterns in the introductory computer science course'. In *ACM SIGCSE Bulletin* (Vol. 32, No. 1, pp. 80–84). ACM.

Renkl, A. & Atkinson, R. K. (2003) 'Structuring the transition from example study to problem solving in cognitive skill acquisition: a cognitive load perspective', *Educational Psychologist*, vol. 38, no. 1, pp. 15–22.

Resnick, M., *et al*., (2009) 'Scratch: programming for all', *Communications of the ACM*, vol. 52, no. 11, pp. 5–15.

Smith, D. C., Cypher, A., & Tesler, L. (2000). 'Programming by example: novice programming comes of age'. *Communications of the ACM*, vol. 43, no. 3, pp. 75–81.

Stachel, J., *et al*., (2013) 'Managing cognitive load in introductory programming courses: a cognitive aware scaffolding tool', *Journal of Integrated Design and Process Science. Computer Science*, vol. 17, no. 1, pp. 37–54.

Sweller, J. (2010) 'Cognitive load theory: recent theoretical advances', in *Cognitive Load Theory*, eds J. L. Plass, R. Moreno & R. Brünken, Cambridge University Press, New York, pp. 29–47.

Sweller, J. (2016) 'Story of a research program', *Education Review/Reseñas Educativas*, vol. 23. URL 1, (2016), *MIT Media Lab*, [online] Available at: https://scratch.mit.edu

Van Merriënboer, J. J. G & Ayres, P. (2005) 'Research on cognitive load theory and its design implications for e-learning', *Educational Technology Research and Development*, vol. 53, no. 3, pp. 5–13.

Weintrop, D. & Wilensky, U. (2015). 'Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs', *11th Annual ACM Conference on International Computing Education Research, ICER 2015*, Association for Computing Machinery, Inc. Omaha, United States, pp. 101–110.

Wing, J. M. (2006) 'Computational thinking', *Communications of the ACM*, vol. 49, no. 3, pp. 33–35.

Wing, J. M. (2008) 'Computational thinking and thinking about computing', *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 366, no. 1881, pp. 3717–3725.

Wolfe, J. M. (2000) *Visual attention*, In Seeing, Academic Press, San Diego.