The Impact of SRA-Programming on Computational Thinking in a Visual Oriented Programming Environment

Fanchamps, Nardie L.J.A.; Slangen, Lou; Specht, Marcus; Hennissen, Paul

DOI
10.1007/s10639-021-10578-0

Publication date
2021

Document Version
Final published version

Published in
Education and Information Technologies

Education and Information Technologies (2021) 26:6479–6498
https://doi.org/10.1007/s10639-021-10578-0

The Impact of SRA-Programming on Computational Thinking in a Visual Oriented Programming Environment

Nardie L. J. A. Fanchamps[1,2] · Lou Slangen[1] · Marcus Specht[3] · Paul Hennissen[1,2]

Received: 28 December 2020 / Accepted: 3 May 2021 / Published online: 28 June 2021

Abstract
Visual programming environments are popular instruments in teaching Computational Thinking (CT) in schools today. Applying Sense-Reason-Act (SRA) programming can influence the development of computational thinking when forcing pupils to anticipate the unforeseen in their computer programs. SRA-programming originates from the programming of tangible robots, but can also be of equal value in visual programming with on-screen output. The underlying rationale is that programming in a visual programming environment using SRA leads to more understanding of the computational concepts addressed, resulting in a higher level of computational skill compared to visual programming without the application of SRA. Furthermore, it has been hypothesised that if pupils in a visual programming environment can anticipate unforeseen events and solve programming tasks by applying SRA, they will be better able to solve complex computational thinking tasks. To establish if characteristic differences in the development of computational thinking can be measured when SRA-programming is applied in a visual programming environment with an on-screen output, we assessed the applicability of SRA-programming with visual output as the main component of the execution of developed code. This research uses a pre-test post-test design that reveals significant differences in the development of computational thinking in two treatment conditions. To assess CT, the Computational Thinking Test (CTt) was used. Results show that when using SRA-programming in a visual programming environment it leads to an increased understanding of complex computational concepts, which results in a significant increase in the development of computational thinking.

Keywords SRA-programming · Computational Thinking · Visual Programming · On-screen Output

✉ Nardie L. J. A. Fanchamps
nardie.fanchamps@fontys.nl

[1] Fontys University of Applied Science, Mgr. Claessenstraat 4, 6131 AJ Sittard, Netherlands

[2] Zuyd University of Applied Science, Nieuw Eyckholt 300, 6419 DJ Heerlen, Netherlands

[3] Delft University of Technology, Mekelweg 4, 2628 CD Delft, Netherlands

1 Introduction

Computational thinking as an important component of Information Communication Technology (ICT) competences which can be developed by means of learning to program (Kennisnet, 2015). Programming works as a means to stimulate the use of Computational Thinking (CT) skills (Edwards, 2005). To stimulate the development of computational thinking, in addition to just learning the basic principles of programming, it is necessary to provide appropriate and directional tasks (Bers et al., 2014). Being able to respond to changes in a task design by means of Sense-Reason-Act (SRA) programming can ensure that users develop computational thinking skills of a higher level (Riedmiller & Gabel, 2007). However, it is still unclear what influence the type of programming environment and the task design used have on this development. It is also unknown whether a visual on-screen programming environment can generate the same performance in comparison with previous research in which SRA-programming was applied in a visual programming environment with physically tangible output by using Lego Mindstorms robots.

There are many different programming languages and environments that are used to let pupils learn tenets of programming. Nowadays many of these are visually oriented environments which offer an easy accessible way to learn programming (Price & Barnes, 2015). Mainly these approaches of visual programming enable pupils to learn the semantics of programming while not having the burden of learning the syntax at the same time, because this is already implemented in drag and drop code blocks (Jost et al., 2014; Navarro-Prieto & Cañas, 2001). Many visual programming environments also have the advantage that the output and effects of code is directly represented as screen output (Chao, 2016).

Visual programming environments come in different appearances. Characteristic for these kinds of environments (e.g. Lego NXT and EV-3, Robomind, Ardublock, etc.) is that they enable novice programmers to gain a user-friendly experience and easy access to the programming paradigm, because the programming syntax is already implemented in visual code blocks (Jost et al., 2014). In primary schools these environments are used to introduce children into programming (Sáez-López et al., 2016; Weintrop & Wilensky, 2015). Their accessibility, simple availability and uncomplicated application is often the reason for using them in the classroom (Repenning, 2017). Visual programming environments are, due to their high imaginative power, ideally suited to teach pupils the underlying concepts of programming (Carlisle, 2009).

In previous research, several authors introduced the SRA-principle which has been identified as an instrumental way of thinking for learning to program robots (Fanchamps et al., 2019; Slangen, 2016; Wong, 2014). SRA can be seen as a didactic instrument to ease the process of programming. When using an SRA-approach whilst programming, there is an initial process in which sensing input detects a state in the (virtual/real) environment (*sense*) and then compares this external state with internal values of the program (*reason*) and finally uses the program to make the necessary inferences on actuators (*act*). SRA-programming

offers a way of conscious thinking in terms of sense-reason-act which supports pupils in solving programming tasks. SRA-programming encourages the effective and efficient use of programming commands. It stimulates pupils to understand the difference between just placing programming commands in one linear sequence versus the programming of parallel structures which inevitably need the use of iterations, conditionals and/or functions. As seen from previous work (Fanchamps et al., 2020), learning to consciously apply SRA contributes to the development of computational thinking. Understanding the principles of SRA does not imply that pupils are always able to recognise and apply these in more complex situations. Research conducted by Caci et al. (2013), Krugman (2004) and Slangen (2016) shows that applying SRA-programming improves pupils' programming skills and abilities to solve tasks of higher complexity. It is also demonstrated when physically tangible robots are programmed through the application of SRA, it leads to a higher level of development of computational thinking skills among primary school pupils, then when physical robots are programmed without the application of SRA (Fanchamps et al., 2019).

Visual programming environments appear to offer excellent possibilities for SRA-programming, making the connection between sensing (input), reasoning and acting (output) visible and comprehensible (Fanchamps et al., 2019; Slangen, 2016; Wong, 2014). Appling SRA is an advanced way of programming that manifests itself in the development of CT-skills (e.g. completion, debugging, sequencing), which has been demonstrated in the programming of physical robots (Fanchamps et al., 2020). It is questionable whether comparable results can be obtained when pupils are provided with a visual programming environment with only on-screen output. Therefore, this research intends to examine the effect of SRA-programming in a visual environment and the impact on computational thinking. Second, the focus is on the influence of applying SRA in a visual programming environment to solve more complex computational thinking tasks.

2 Theoretical framework

In this research we are specifically interested in the impact of applying SRA-programming in visual programming environments on computational thinking. We also want to distinguish if applying SRA results in a better capability to solve more complex computational thinking tasks.

From our research we know that the application of SRA when programming tangible robots affects the development of computational thinking skills (Fanchamps et al., 2019). We also know that a dynamic environment in which unforeseen situations occur, in which pupils can no longer rely on linear, sequential programming to successfully solve programming tasks, it requires the application of SRA-thinking (Fanchamps et al. submitted). The construct of SRA-programming can be described as a process in which external, sensor-based observations (*sense*) are fed into a microprocessor so that these observations can be compared with internal, preset conditions (*reason*) from which the execution of desired programming actions (*act*) can be derived. The ability to anticipate changing conditions in the task design

through sensor-based observations requires a different programming approach compared to linear solutions (Dragone et al., 2005). SRA-programming involves the functional understanding and application of complex programming concepts such as loops, conditionals and functions (Slangen, 2016). Sensor-based programming is known as a smart way of programming (Johansson & Balkenius, 2006).

To understand the construct of SRA-programming and being able to apply it effectively, more insight is needed into what underlying complex programming concepts, such as iterations, functions and conditionals do. Programming making functional use of loops, the use of sensors and being able to program based on cause/effect relationships is fundamental for SRA-programming. It is precisely the application of SRA that ensures that the user can discover in a functional and accessible manner, what advantages the application of complex programming concepts can provide. This in a direct comparison with linear programming in which changes in the task design cannot be anticipated ad hoc via the constructed program itself. In that case, a manual adjustment in the program itself is always necessary and has to be conducted by the user. In contrast, in SRA-programming a triggered adaptation is provided for, where changing external conditions are constantly compared with pre-set values. Based on these changing conditions, the program itself then decides which alternative to follow. For example: if a programmed artefact only needs to initiate an evasive command at a distance of 10 cm from an obstacle, or if a colour sensor always needs to initiate a specific action when a predetermined colour is detected. In such cases, a specific repeat and "if–then-else" condition is used. In these kind of situations SRA-programming comes into play.

Slangen (2016) and Fanchamps et al. (submitted) investigated the functional application of the SRA-approach in programmable robotic environments. Both came to the conclusion that pupils understood tenets of programming better when applying SRA. Despite the fact that complex programming concepts, such as the “repeat”, “repeat until, “if”, “if–then”, “if–then-else”, “while” and “simple functions” were understood, they were not applied as a matter of course in new programming assignments. Román-González et al. (2017) addressed these concepts in their Computational Thinking Test (CTt) in order to make computational thinking measurable. In this test computational thinking is operationalised by a number of concepts, namely loops, conditionals, functions, nesting and the required computational thinking tasks debugging, completion, sequencing.

Computational thinking is a way of tackling and solving problems making use of computer science concepts and it primarily involves the ability to reason, plan and solve problems (Wing, 2006). It refers to operationalised concepts such as parallel thinking, pattern recognition, completion, debugging, sequencing, and abstract reasoning needed to approach a problem systematically (Basawapatna et al., 2011; Lee et al., 2011). Computational thinking involves the process in which problem definition, solution expression, and implementation with evaluation occurs recurrently in the process of programming (Yadav et al., 2016), and can contribute to understand and to solve complex programming problems (Voskoglou & Buckley, 2012). Application of the SRA-approach can ensure that conceptual understanding develops, allowing a higher level of computational thinking to be achieved (Basu et al., 2016). In our exploratory research

we therefore investigate to what extent the application of SRA during visual programming influences the development of computational thinking (i.e. completion, debugging, sequencing).

Visual programming environments with an on-screen output are very suitable for use in primary education. Such environments can be characterized by the use of underlying syntax and semantics to support the act of programming (Weintrop & Wilensky, 2015). Visual programming environments offer the possibility to learn how to program in an accessible way by dragging, dropping and connecting manipulatable code blocks into a worksheet on screen to create a program (Asad et al., 2016). These code blocks can contain iterations, conditions, functions and routines using parameters, variables and operators (Price & Barnes, 2015). Often visual programming environments are a first introduction to programming for pupils (Jost et al., 2014). Many of these environments offer the possibility to vary the level of difficulty because they have a build-up in complexity. This allows the application to connect to the starting level of each individual user. Level differentiation can also be applied by enabling or disabling options which makes it possible that a new challenge can be realised for each user at their own level, resulting in maximum learning efficiency (Sapounidis et al., 2015). When used in this way, an adapted environment can be provided that meets the pre-experience, age level and current skill level of pupils (Repenning, 2017). Despite the fact that a visual environment seems eminently suitable for learning how to program, it is questionable whether pupils can use such an environment to achieve programming skills of higher complexity (Kaučič & Asič, 2011). Our previous research indicates that pupils, when programming robots have difficulty applying the principles of SRA despite understanding these principles (Fanchamps et al., 2019). However, when pupils find themselves in a situation where the programming task only can be solved through an SRA-approach, they tend to use complex programming concepts in a way as operationalized in computational thinking.

Building on the theoretical exploration above, we assume a correlation between the SRA-approach in a visual programming environment with an impact on computational thinking. Therefore, our conceptual model in Fig. 1 provides an overview of relationships and interconnections between the independent and the dependent variables.

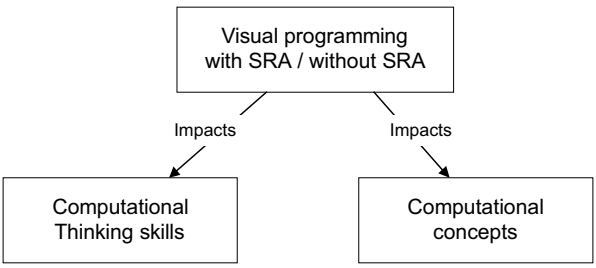

Fig. 1 Schematic representation of the conceptual model

3 Research question, sub-questions and hypothesis

Based on preliminary studies and the literature research, our main research question is: "What is the effect of SRA-programming in a visual programming environment on computational thinking and specifically on the computational concepts addressed?".

This main research question leads to the next sub-questions:

1. Can SRA-programming in a visual programming environment with an on-screen output be of added value concerning the development of computational thinking?
2. What impact on computational thinking can be achieved by integrating SRA-programming in a visual programming environment?
3. What is the influence of SRA-programming in a visual programming environment on complex computational concepts?

These sub-questions result in the following hypotheses:

1. The actual application of SRA-programming in a visual programming environment leads to a more successful solution of computational thinking tasks.
2. Applying SRA-programming in a visual programming environment leads to a higher level of development of computational thinking skills, compared to programming in a visual environment without the use of SRA.
3. The actual application of SRA-programming in a visual programming environment with on-screen output enables a higher understanding of complex computational concepts.

4 Method

In this research an intervention study (Creswell 2008) is used to identify the effect of SRA-programming on the development of computational thinking. To determine whether using SRA in two programming conditions (visual programming with or without the use of SRA) can make a difference in results, pupils use Bomberbot© as a visual programming environment with on-screen output to construct and to solve programming tasks.

As a pre- and post-measurement, the Computational Thinking Test (CTt) was applied. Quantitative data has been collected in order to answer the research question, sub-questions and hypotheses. In order to investigate the effect of the intervention, we used a pre-test post-test design as illustrated in Fig. 2. This includes (a) pre-measurement of computational thinking skills, (b) a basic instruction on how Bomberbot works, (c) a programming intervention in two variants and (d) post-measurement of computational thinking skills.

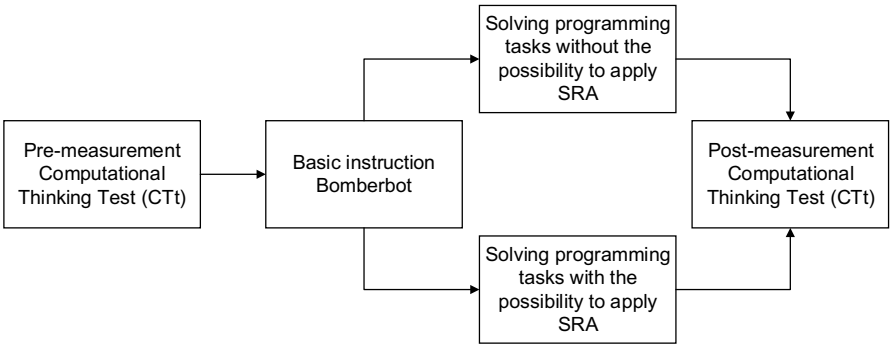

Fig. 2 Research design

4.1 Participants

This research was conducted among pupils from grade 5[1] of a primary school ($N=30$) in the Netherlands from which, at random, an experimental group ($n=15$) and a control group ($n=15$) of equal size were composed. Apart from using applications such as Word, PowerPoint, Excel and working with games and apps, pupils have no specific programming experience and have never used Bomberbot before.

4.2 Materials

To answer our research questions we used the visual programming environment Bomberbot to let pupils learn the concepts of programming and from which, by application, we want to deduce what the effect of SRA-programming is. It should be noted that the term SRA comes from the physical world of robotics and refers to the use of material sensors and actuators. In the visual environment applied sensors and actuators are by proxy virtual sensors and actuators. Bomberbot is an on-screen, game based and visually oriented platform that offers children the opportunity to learn programming in a playful, user-friendly way. In Bomberbot a virtually programmable robot is required to perform tasks such as: collecting stars, smashing gems or obstacles, opening treasure chests by means of a colour code that can be deciphered, etc. To construct a program in Bomberbot, the necessary programming commands can be dragged from the library on the left and dropped into the main program. The available commands can be accessed using the scroll menu, as displayed in Fig. 3. The library contains a selection of pictographical commands, such as: forward, backward, turn left, turn right, hit with hammer, jump, repeat, if/then, function (F1), etc. In the main program, the commands can be sequenced and arranged in the desired order. For the application of functions, as displayed in Fig. 4, an additional function worksheet is available below the main program. It can contain

[1] In this publication we use the UK grade level system to indicate the research population. Grade 5 in the UK corresponds with the Dutch "group 7".

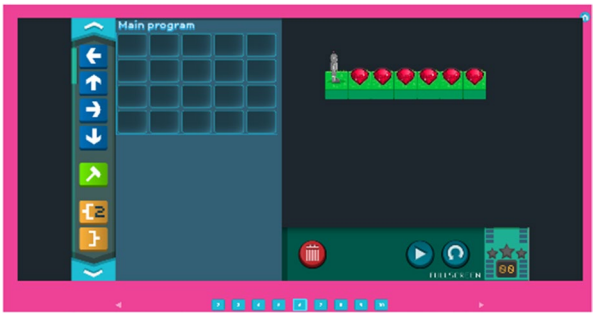

Fig. 3 Basic display Bomberbot© with library and main program

the commands that are called upon when the function button is placed in the main program. To apply the constructed program the play button is pressed. The repeat function allows to run the task again from the start. The number display with the three stars above it indicates how efficient the constructed program is designed and can therefore be used as an incentive. The bin can be used to delete programming commands.

By using Bomberbot, pupils learn to apply fundamental concepts and skills of programming and can use this programming environment to get a grip on what iterations, routines, functions and conditionals are and how they should be used. Because Bomberbot is designed to provide direct feedback, pupils can judge for themselves whether the constructed program is correct or not, allowing them to learn in a self-correcting way. Within this environment there is an increase in difficulty and complexity.

To determine the level of computational thinking skills in the pre- and post-measurement, we used the validated Computational Thinking Test (CTt) (Román-González et al., 2017). In this test computational thinking is operationalised through a number of concepts, namely loops, conditionals, functions, nesting and the required computational thinking tasks debugging, completion, sequencing. Pupils completed this test individually. The test contains 28 items addressing the computational concepts of basic directions, loops (repeat times, repeat until), conditionals (if-simple, if-else-complex, while) and functions (simple functions, functions with parameters). The existence or non-existence of nesting is determined. The computational task/skill required

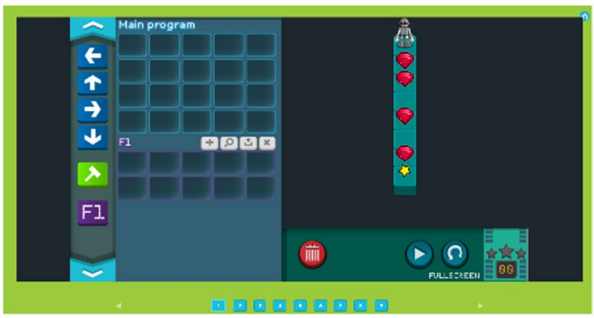

Fig. 4 Display Bomberbot© with additional function worksheet F1

Fig. 5 Linear solution Bomberbot©

(completion, debugging, sequencing) to provide the right solution for each of the 28 test-items can also be derived. To determine the reliability of the scale, we calculated Cronbach's alpha. It should be noted that a value for Cronbach's alpha of 0.70 is considered an acceptable reliability factor (Santos, 1999). As a characteristic, the developers of this instrument indicate that for grade 5 ($N = 176$) Cronbach's alpha should be $\alpha = 0.721$ (Román-González et al., 2017). We measured for Cronbach's alpha $\alpha = 0.67$ whereby this value almost complies the required level of internal consistency for our scale with this particular sample. From this we can conclude that, despite the low value for N, the internal reliability of the adapted instrument used is sufficient and performs as reported by the original authors. The measurement results will therefore be used as such.

As an intervention, we used Bomberbot© as a visual programming environment with an on-screen-based output. Bomberbot can be used without the application of SRA. To make this possible, the application of loops, conditionals and functions can be disabled. In this condition linear, sequential programming must be used to solve programming tasks. Linear, sequential programming means that the user of the programming environment does not use any programming commands based on parallel processes such as iterations, conditionals and functions to solve programming tasks. For the other condition, Bomberbot can be used for solving programming tasks with or without the application of SRA.

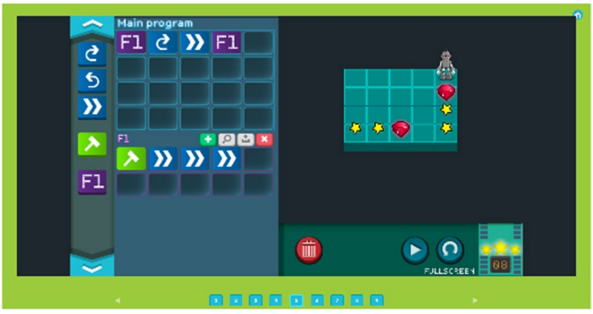

Fig. 6 SRA-solution Bomberbot©

To solve SRA-programming tasks, all the options and applications of loops, conditionals and functions within Bomberbot can be used, but the more simple linear approach is still available. As an example, we include two screen shots of Bomberbot in which a virtually programmable robot has to collect stars and smash rubies. In order for the robot to perform these tasks, the users must drag the appropriate programming commands into a workspace. They can use linear, sequential programming commands or by the SRA-approach the more complex commands such as loops, conditionals and functions. The tasks contain an incentive to stimulate to program as efficiently as possible within the given conditions by earning points displayed as stars. Figure 5 shows that the assignment is solved by linear programming. Figure 6 shows that the same assignment is solved via the application of SRA by making use of functions.

4.3 Procedure

As displayed in the research design, we defined two subgroups who received the same basic introduction on how to use Bomberbot operationally. The supervisor provided the instruction on how to use Bomberbot. We have chosen to divide pupils into pairs, because in this way we have an opportunity to follow reasoning processes by means of the learning dialogue. Consecutively, pupils of both subgroups completed five one-hour programming sessions to solve ten programming missions containing 15 programming tasks each in order to demonstrate the solutions found. The supervisor only helped when there were technical problems. Bomberbot offers the possibility to apply an SRA-approach by means of using virtual sensors and specific programming concepts such as iterations, conditionals and functions available or not available within programming tasks of similar complexity. In Bomberbot the programming tasks consist of moving a virtual robot from a starting point to an end point by selecting and combining relevant commands and executing them. Along the way, the robot has to perform additional tasks, such as jumping on the spot, hammering gems, making a half turn, opening a box, removing objects, etc. The robot also encounters other/unforeseeable circumstances in which a) the control group controls the robot through its standard, linear program and b) the experimental group has the choice of controlling the robot using either the linear approach or the SRA-approach using virtual sensors and programming concepts to anticipate these changing conditions.

5 Results and Data-analysis

The main research question, “What is the effect of SRA-programming in a visual programming environment on computational thinking and specific computational concepts addressed?”, is answered by analysing the means of the

dichotomous variables and T-test analysis is used to examine sub-questions and to confirm or reject hypotheses. The results of the pre- and post-measurement of the Computational Thinking Test (CTt) are entered into SPSS for quantitative data analysis. Differences in values are determined by comparing the means. Using cross tabs, a shift between pre- and post-measurement is made visible. In all statistical analyses a significance level of 5% ($p = \leq .\ 05$) is assumed.

The nature of the data meets the conditions for the assumption of normality and asserts that the distribution of sample means (across independent samples) is normal. It has been tested whether the assumptions of homogeneity of variances have been violated ($p \leq 0.05$). Degrees of freedom are calculated and the bootstrapping procedure has been applied to re-estimate the standard error of the mean difference. The confidence interval is studied to assess the difference between means and to determine whether the value "0" is in the confidence interval. The value for the extent of the effect (Pearson's r) has been calculated (indicating that the effect size is low if the value of r varies around 0.1, medium if r varies around 0.3, and large if r varies more than 0.5). The substantial effect of a standard deviation difference between two groups (Cohen's d) was also determined (it should be noted that $d = 0.2$ can be considered a 'small' effect size, 0.5 stands for a 'medium' effect size, 0.8 for a 'large' effect size and where any value above 1.4 is considered a very large effect) (Field, 2013).

5.1 Differences in the level of development of computational thinking skills

Table 1 shows the data for the level of development of computational thinking skills comparing both groups.

Comparing the means between the pre-measurement and the post measurement firstly show that pupils who applied SRA-programming in the visual programming environment on the Computational Thinking Test (CTt) a) solved more computational thinking items successfully, and b) indicated higher values regarding sequencing, completion and debugging for the required task application. Secondly, the influence of SRA-programming showed c) higher ratings on the understanding of the working principles of loops, conditionals and functions, and d) pointed out whether or not to apply nesting. This was deduced from the data whereby the group that programmed with SRA in the post-measurement shows a higher average score (M), a lower standard deviation (SD) and less spread in the measured values (*range*). Based on these measurements, it is reasonable to assume that programming with SRA in the visual programming environment is the cause of this increase.

Table 1 Differences in level of development of Computational Thinking Skills ($N=30$)

	Pre-test group without SRA ($n=15$)					Post-test group without SRA ($n=15$)			
Variable	*M*	*SD*	Range	*Mdn*	Variable	*M*	*SD*	Range	*Mdn*
Total (28)	0.44	0.10	0.39 – 0.50	0.43	Total (28)	0.45	0.13	0.38 – 0.55	0.43
Loops	0.40	0.12	0.24 – 0.64	0.40	Loops	0.40	0.13	0.20 – 0.68	0.40
Conditionals	0.35	0.15	0.11 – 0.58	0.36	Conditionals	0.34	0.12	0.08 – 0.50	0.33
Functions	0.43	0.27	0.00 – 1.00	0.25	Functions	0.47	0.27	0.00 – 0.75	0.50
Nesting	0.36	0.13	0.16 – 0.53	0.37	Nesting	0.36	0.12	0.16 – 0.58	0.37
CT-skill Completion	0.51	0.23	0.11 – 0.78	0.56	CT-skill Completion	0.50	0.13	0.33 – 0.67	0.56
CT-skill Debugging	0.49	0.21	0.20 – 1.00	0.40	CT-skill Debugging	0.40	0.24	0.00 – 0.80	0.40
CT-skill Sequencing	0.44	0.14	0.21 – 0.64	0.50	CT-skill Sequencing	0.43	0.19	0.14 – 0.86	0.50
	Pre-test group with SRA ($n=15$)					Pre-test group with SRA ($n=15$)			
Variable	*M*	*SD*	*Range*	*Mdn*	Variable	*M*	*SD*	*Range*	*Mdn*
Total (28)	0.41	0.12	0.30 – 0.50	0.39	Total (28)	0.67	0.07	0.61 – 0.70	0.64
Loops	0.38	0.13	0.20 – 0.60	0.40	Loops	0.62	0.09	0.52 – 0.80	0.60
Conditionals	0.31	0.14	0.14 – 0.67	0.28	Conditionals	0.63	0.11	0.42 – 0.78	0.67
Functions	0.23	0.15	0.00 – 0.50	0.25	Functions	0.83	0.15	0.50 – 1.00	0.75
Nesting	0.29	0.12	0.11 – 0.58	0.32	Nesting	0.64	0.09	0.53 – 0.79	0.63
CT-skill Completion	0.40	0.14	0.22 – 0.56	0.44	CT-skill Completion	0.69	0.12	0.44 – 0.89	0.67
CT-skill Debugging	0.45	0.29	0.00 – 0.80	0.40	CT-skill Debugging	0.61	0.26	0.20 – 1.00	0.60
CT-skill Sequencing	0.42	0.13	0.29 – 0.71	0.43	CT-skill Sequencing	0.68	0.12	0.43 – 0.93	0.64

Variable = measurable value; Total = number of items correct CT−test; Computational concept addressed = loops, conditionals, functions, nesting; completion = completed by CT; debugging = reformulate problems; sequencing = sequence; *M* average; *SD* standard deviation; range = spread in measurement, *Mdn* = median

5.2 Selection of SRA-programming

An inventory on how often pupils in the group that had SRA at their disposal actually applied SRA-programming is displayed in Table 2. Pupils could decide for themselves whether or not to solve tasks using SRA. It can be deduced that in approximately half of all programming missions an application of SRA has been selected. Where the application of SRA-programming has not been selected, pupils opted for solving programming tasks making use of linear, sequential programming structures.

Table 2 Selection SRA-programming

Pupils	*Mission 1 loops*	*Mission 2 conditionals*	*Mission 3 functions*	*Total*
Pair 1	10 out of 13 tasks	9 out of 10 tasks	6 out of 16 tasks	25 out of 39 tasks
Pair 2	12 out of 13 tasks	5 out of 10 tasks	2 out of 16 tasks	19 out of 39 tasks
Pair 3	11 out of 13 tasks	8 out of 10 tasks	3 out of 16 tasks	22 out of 39 tasks
Pair 4	13 out of 13 tasks	5 out of 10 tasks	0 out of 16 tasks	18 out of 39 tasks
Pair 5	10 out of 13 tasks	6 out of 10 tasks	5 out of 16 tasks	21 out of 39 tasks
Pair 6	10 out of 13 tasks	7 out of 10 tasks	2 out of 16 tasks	19 out of 39 tasks
Pair 7	12 out of 13 tasks	6 out of 10 tasks	4 out of 16 tasks	22 out of 39 tasks
Pair 8	11 out of 13 tasks	2 out of 10 tasks	1 out of 16 tasks	14 out of 39 tasks

Pair = pupils divided in pairs; Mission 1 = application of the variable loops; Mission 2 = application of the variable conditionals; Mission 3 = application of the variable functions; Total = SRA applied in relation to the total number of missions.

5.3 Impact SRA on solving computational thinking tasks

Comparing the different t-tests carried out on all the variables concerned shows that, in general, there are statistically significant differences regarding the correct number of solved items from the Computational Thinking Test (CTt) and the computational concept addressed and that the characteristics of computational thinking are increasing as a result of SRA-programming (Table 3). Since the value "0" is not within the confidence interval for each variable, this confirms the assumption that applying SRA-programming in a visual programming environment provides a significantly higher level of solving programming tasks, resulting in a higher level of computational thinking skill. Calculating the size of the effects on the independent t-test (effect-size) also shows that there are large effects measurable.

Table 3 T-test analysis comparing visual programming based on SRA usage

Variable	*t*	*df*	*p*	*CI*	*d*
Total (28)	5.93	20.92	0.000	0.14 – 0.30	2.18
Loops	5.42	28.00	0.000	0.14 – 0.31	1.94
Conditionals	6.91	27.60	0.000	0.20 – 0.38	2.52
Functions	4.63	22.51	0.000	0.20 – 0.53	1.66
Nesting	7.17	28.00	0.000	0.20 – 0.36	2.64
CT-skill Completion	4.02	28.00	0.000	0.09 – 0.28	1.51
CT-skill debugging	2.36	28.00	0.013	0.03 – 0.40	0.85
CT-skill Sequencing	4.18	28.00	0.000	0.12 – 0.36	1.57

Variable measurable value; Total = number of items correct CT−test; Computational concept addressed = loops, conditionals, functions, nesting; completion = completed by CT; debugging = reformulate problems; sequencing = sequence; t = t−value; df = degrees of freedom; p = p−value; CI = confidence interval; d = effect size

6 Conclusion

An interpretation of the analysed data shows that there is a significant increase in the level of computational thinking skills through the application of SRA-programming, using a visual programming environment. Significant yields are measured on all variables present in this research. The extent of the effect by applying SRA-programming is large to very large. The findings in this research show a high impact on computational thinking due to SRA-programming.

In order to find out whether SRA-programming has a significant increase in pupils' understanding of complex computational concepts, resulting in a higher level of computational thinking skills, we used the results of the control group that did not have SRA-programming at its disposal in a comparison with the group that could use SRA-programming to solve programming tasks. From the results obtained we hardly see any development in CT-skills when pupils program in a purely linear environment. We can conclude from this that it cannot be stated straightforwardly that programming automatically leads to more skills of programming. The group which had SRA at its disposal, used SRA to solve programming tasks in about half of all assignments. This choice was due to the fact that pupils could decide by themselves to solve the missions either with linear programming or SRA-programming. Our research made visible that, when there was a motivational intent or a trigger to look for a more thoughtful and efficient solution, SRA-programming was applied. Where pupils actually opted for an application of SRA we see, in a comparison with the linear approach, through a better understanding of complex computational concepts, a significant increase in the development of computational thinking skills. The extent of the effect by applying SRA-programming is large to very large. The findings in this research indicate a high impact on computational thinking due to SRA-programming. The hypotheses that the actual application of SRA-programming in a visual programming environment leads to a more successful solution of computational thinking tasks can be confirmed. The sub-question whether SRA-programming in a visual programming environment with on-screen output is of added value related to a development in computational thinking can be positively endorsed.

It is often claimed that programming leads to the development of computational thinking skills. It is questionable whether this can be stated in such a generic way. The suspicion is that the nature of the task and the programming environment helps to determine whether there is a development in CT-skills. If we look at the analysed data of pupils who only had the linear environment as their task, we can say that there is hardly any development measureable. However, in the environment where pupils did have access to SRA, we see that only the pupils that actually choose to apply SRA show a significant increase on computational thinking. From these results, the hypothesis can be confirmed that applying SRA-programming in a visual programming environment leads to a higher level of development of computational thinking skills compared to programming in a visual environment without the use of SRA.

The measurements in our research indicate that the application of SRA-programming in a visual programming environment with on-screen output allows pupils a

better grasp of the computational concepts addressed. The effect, when pupils are allowed to choose whether they prefer a linear approach as the nature of the task versus the effect evoked when pupils choose an SRA task solution, is remarkably visible. In the latter case, pupils generally score significantly better on complex computational concepts and CT-skills in the post-test. The choice to opt for SRA-programming leads to a significant better solution of CT-tasks. This means that when pupils opt for an SRA-approach, they are better able to get to grips with the underlying concepts that lead to a higher level of CT-skills. The hypothesis that the actual application of SRA-programming in a visual programming environment with on-screen output enables a higher understanding of complex computational concepts can be confirmed.

Overall concluding, we created an environment in which two conditions were used in which one group of pupils programmed linearly and the other group could choose to apply/not to apply SRA-programming. With the help of an independent Computational Thinking Test (CTt) it appears that pupils who have programmed in the SRA condition solve CT tasks better. From this we can deduce that the explicit application of the SRA-approach also leads to acquire more understanding of complex computational concepts. The absence of SRA leads to no gain in computational thinking skills. If there is no explicit focus on the use of computational concepts, one can solve simple programming problems, but this has no effect on the further understanding of complex computational concepts. This becomes visible through the results of the Computational Thinking Test (CTt). We suspect that pupils who already have previous experience of programming with loops, conditionals and functions are more likely to use them. In Bomberbot pupils are also invited by earning stars to find and use the most efficient way of programming. This can be seen as a stimulus for pupils to apply SRA. By actually establishing the relationship between applying SRA and constructing the most efficient program, pupils show that they understand the underlying computational thinking concepts (e.g. loops, conditionals, functions).

The freedom of choice, whether or not to apply SRA learning the underlying insights of the concepts by carrying out the tasks, triggers the discovery of the underpinning functionality. This leads to a self-learning understanding of the underlying concepts, which in turn leads to a further development of CT skills. What makes this plausible is that pupils who had SRA at their disposal choose to opt for the application of SRA in half of all programming tasks and not for the other half. Decisive for the eventual choice to opt for SRA is that pupils are challenged to find a qualitatively better solution to solve programming tasks. Pupils who choose the qualitatively better solution score better on the CT test based on that.

7 Discussion

The aim of this research is to find answers to the question what effect SRA-programming in a visual programming environment with on-screen output has on computational thinking and specific computational concepts addressed.

First of all, there are claims that programming improves children's computational thinking skills. We would like to take a more nuanced view of that statement. After all, if pupils are offered a programming environment that is limited to linear programming, this hardly leads to a growth in CT-skills (e.g. sequencing, debugging, completion). The nature of the task can determine whether or not a development in CT skills is shown. If pupils are given the opportunity to use SRA-programming, we see that pupils are able to fathom the application of computational concepts addressed which leads to a higher level of CT-skills.

The results obtained from our research demonstrate that a visual programming environment with on-screen output can integrate SRA-programming concepts and show foreseen effects of SRA-programming on computational thinking. This is in line with claims made by Jost et al. (2014), Papadakis et al. (2016), Repenning (2017) and Strawhacker and Bers (2015) who state that visual/on-screen oriented programming environments provide an easy accessible way with a low-threshold, functional use in primary school education whereby pupils acquire functional programming skills of a higher level. Moreover, our research confirms that applying SRA in a visual programming environment can create a higher level of computational thinking.

The results show that by applying SRA-programming pupils demonstrate a subsequent development of computational thinking skills. In addition, our results show that, due to the impact of the SRA-approach, pupils can solve complex computational thinking tasks and can ensure that pupils get a better grip on complex concepts of programming.

This research contributes to the theory building of computational thinking and programming education in primary schools. Visual programming environments make pupils use computer technology to examine and solve problems systematically and in a creative and interactive way with sufficient possibilities for differentiation. It was claimed in a similar way by Asada et al. (1999) for a tangible robotics programming environment with unforeseen events, pupils need to anticipate such changing circumstances in a visual, on-screen programming environment through their programming actions. SRA-programming is proving to be an added value in this respect and can maybe be seen as an unidentified characteristic of computational thinking. As stated by Bocconi et al. (2016), we are interested in what other cognitive effects can be made measurably transferring computational thinking skills to other knowledge domains.

This research also contributes to the way in which programming in education can be operationalised. Based on the significant results, it can be concluded that the SRA-approach is a promising concept for obtaining computational thinking skills in a more profound and meaningful way. For this it has to be made sure that pupils do work with complex concepts otherwise there will be no growth.

7.1 Limitations and follow-up research

With regards to the results of this research, several reservations can be made. Reasons for this may be that pupils in their regular primary school curriculum have

already continued to work with programming environments or that there is a regular development of pupils over time due to their standard educational program.

This research was carried out by making use of the visual programming environment Bomberbot with an on-screen output. In order to be able to generalize more from our research, this research should be replicated with other visual programming environments with an on-screen output. It should also be examined whether the nature of the programming task affects the outcome and the learning effects.

In subsequent research it is valuable to evaluate whether SRA-programming in a visual programming environment with on-screen output shows the same yields on the development of computational thinking skills as programming in a visual programming environment with a physical perceptible and tangible output. It is also interesting to investigate whether the yield of programming with a tangible output improves when pupils firstly operate with programming environments with only an on-screen output.

It is important to consider that initiating, supporting and supervising programming activities requires specific competences from the teacher. However, a teacher should be well equipped to adequately facilitate and guide such activities. In this research, the researcher acted also as the qualified, supporting supervisor. In a subsequent study, it is valuable to investigate whether comparable returns can be found when a regular, competent teacher takes on the role of supervisor and the researcher is present purely as an observer.

This research was conducted with a relatively small group of respondents from which pairs were composed. The small amount of the research sample, as well pupils working in pairs, could have been influential to our findings. In order to be able to generalize from our findings, it is recommended to repeat this research with a larger number of respondents.

Acknowledgements The authors would like to thank Bomberbot Netherlands for making the programming environment available and for their cooperation.

Authors' contributions All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by Nardie Fanchamps. The first draft of the manuscript was written by Nardie Fanchamps and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Data availability Can be requested from the first author.

Code availability Retrievable on request from Bomberbot© with personal login code.

Declarations

Ethics approval The Ethical research board (cETO) of the Open University of the Netherlands has assessed the proposed research and concluded that this research is in line with the rules and regulations and the ethical codes for research in Human Subjects (reference: U2019/01324/SVW).

Consent to participate Written informed consent was obtained from the parents of the individual participants.

Consent to publish The parents of the individual participants consented the data obtained to be published.

Conflicts of interest/Competing interests The authors declare that they have no conflict of interest/no competing interests.

References

Asad, K., Tibi, M., & Raiyn, J. (2016). Primary School Pupils' Attitudes toward Learning Programming through Visual Interactive Environments. *World Journal of Education, 6*(5), 20–26. https://doi.org/10.5430/wje.v6n5p20

Asada, M., Kitano, H., Noda, I., & Veloso, M. (1999). RoboCup: Today and tomorrow—What we have learned. *Artificial Intelligence, 110*(2), 193–214

Basawapatna, A., Koh, K. H., Repenning, A., Webb, D. C., & Marshall, K. S. (2011). *Recognizing computational thinking patterns.* Paper presented at the Proceedings of the 42nd ACM technical symposium on Computer science education.

Basu, S., Biswas, G., Sengupta, P., Dickes, A., Kinnebrew, J. S., & Clark, D. (2016). Identifying middle school students' challenges in computational thinking-based science learning. *Research and Practice in Technology Enhanced Learning, 11*(1), 13. https://doi.org/10.1186/s41039-016-0036-2

Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education, 72*, 145–157. https://doi.org/10.1016/j.compedu.2013.10.020

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., Engelhardt, K., Kampylis, P., et al. (2016). *Developing Computational Thinking in Compulsory Education.* https://doi.org/10.2791/792158

Caci, B., Chiazzese, G., & D'Amico, A. (2013). Robotic and virtual world programming labs to stimulate reasoning and visual-spatial abilities. *Procedia-Social and Behavioral Sciences, 93*, 1493–1497

Carlisle, M. C. (2009). Raptor: a visual programming environment for teaching object-oriented programming. *Journal of Computing Sciences in Colleges, 24*(4), 275–281

Chao, P.-Y. (2016). Exploring students' computational practice, design and performance of problem-solving through a visual programming environment. *Computers & Education, 95*, 202–215. https://doi.org/10.1016/j.compedu.2016.01.010

Dragone, M., O'Donoghue, R., Leonard, J. J., O'Hare, G., Duffy, B., Patrikalakis, A., et al. Robot soccer anywhere: achieving persistent autonomous navigation, mapping, and object vision tracking in dynamic environments. In *Opto-Ireland 2005: Photonic Engineering, Dublin, Ireland, 2005* (Vol. 5827, pp. 255–265): International Society for Optics and Photonics.https://doi.org/10.1117/12.608404.

Edwards, S. (2005). Identifying the factors that influence computer use in the early childhood classroom. *Australasian Journal of Educational Technology, 21*(2), https://doi.org/10.14742/ajet.1334.

Fanchamps, N., Slangen, L., Hennissen, P., Specht, M. (2019). The Influence of SRA Programming on Algorithmic Thinking and Self-Efficacy Using Lego Robotics in Two Types of Instruction. *International Journal of Technology and Design Education*, 1-20, https://doi.org/10.1007/s10798-019-09559-9.

Fanchamps, N., Slangen, L., Specht, M., & Hennissen, P. (submitted). The effect on computational thinking using SRA programming when anticipating changes in a dynamic problem environment. *IEEE Transactions on Learning Technologies*, 18.

Fanchamps, N., Specht, M., Hennissen, P., & Slangen, L. (2020). The effect of teacher interventions and SRA robot programming on the development of computational thinking. In S.-C. Kong, & H. Abelson (Eds.), *International conference on computational thinking education 2020, Hongkong, 2020* (Vol. CTE2020, pp. 5). Springer

Field, A. (2013). *Discovering statistics using IBM SPSS statistics*: sage.
Johansson, B., & Balkenius, C. An experimental study of anticipation in simple robot navigation. In *Workshop on Anticipatory Behavior in Adaptive Learning Systems, 2006* (pp. 365–378): Springer. https://doi.org/10.1007/978-3-540-74262-3_20.
Jost, B., Ketterl, M., Budde, R., & Leimbach, T. (2014). *Graphical programming environments for educational robots: Open roberta-yet another one?* Paper presented at the IEEE International Symposium on Multimedia.
Kaučič, B., & Asič, T. (2011). *Improving introductory programming with Scratch?* Paper presented at the 2011 Proceedings of the 34th International Convention MIPRO.
Kennisnet (2015). Computing-onderwijs in de praktijk - Wat kunnen we leren van de Britten? (pp. 81). Kennisnet.
Krugman, M. (2004). *Teaching behavior based robotics through advanced robocamps.* Paper presented at the 34th Annual Frontiers in Education, 2004. FIE 2004.
Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., et al. (2011). Computational thinking for youth in practice. *acm Inroads, 2*(1), 32–37, https://doi.org/10.1145/1929887.1929902.
Navarro-Prieto, R., & Cañas, J. J. (2001). Are visual programming languages better? The role of imagery in program comprehension. *International Journal of Human-Computer Studies, 54*(6), 799–829. https://doi.org/10.1006/ijhc.2000.0465
Papadakis, S., Kalogiannakis, M., & Zaranis, N. (2016). Developing fundamental programming concepts and computational thinking with ScratchJr in preschool education: a case study. *International Journal of Mobile Learning and Organisation, 10*(3), 187–202
Price, T., & Barnes, T. (2015). *Comparing textual and block interfaces in a novice programming environment.* Paper presented at the Proceedings of the eleventh annual international conference on international computing education research.
Repenning, A. (2017). Moving beyond syntax: Lessons from 20 years of blocks programing in AgentSheets. *Journal of Visual Languages and Sentient Systems, 3*(1), 68–89. https://doi.org/10.18293/VLSS2017-010
Riedmiller, M., & Gabel, T. On experiences in a complex and competitive gaming domain: Reinforcement learning meets robocup. In *2007 IEEE Symposium on Computational Intelligence and Games, 2007* (pp. 17–23): IEEE.
Román-González, M., Pérez-González, J.-C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior, 72*, 678–691. https://doi.org/10.1016/j.chb.2016.08.047
Sáez-López, J.-M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education, 97*, 129–141. https://doi.org/10.1016/j.compedu.2016.03.003
Santos, J. R. A. (1999). Cronbach's alpha: A tool for assessing the reliability of scales. *Journal of Extension, 37*(2), 1–5
Sapounidis, T., Demetriadis, S., & Stamelos, I. (2015). Evaluating children performance with graphical and tangible robot programming tools. *Personal and Ubiquitous Computing, 19*(1), 225–237. https://doi.org/10.1007/s00779-014-0774-3
Slangen, L. (2016). *Teaching Robotics in Primary School*. Eindhoven University of Technology.
Strawhacker, A., & Bers, M. U. (2015). "I want my robot to look for food": Comparing Kindergartner's programming comprehension using tangible, graphic, and hybrid user interfaces. *International Journal of Technology and Design Education, 25*(3), 293–319. https://doi.org/10.1007/s10798-014-9287-7
Voskoglou, M. G., & Buckley, S. (2012). Problem solving and computational thinking in a learning environment. *Egyptian Computer Science Journal, 36*(4), 18
Weintrop, D., & Wilensky, U. (2015). *To block or not to block, that is the question: students' perceptions of blocks-based programming.* Paper presented at the Proceedings of the 14th International Conference on Interaction Design and Children, Medford, MA, USA.
Wing, J. M. (2006). Computational Thinking. *Communications of the ACM, 49*(3), 33–35. https://doi.org/10.1145/1118178.1118215
Wong, L. L. (2014). *Rethinking the Sense-Plan-Act Abstraction: A Model Attention and Selection Framework for Task-Relevant Estimation.* Paper presented at the Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence, Quebec, Canada.

Yadav, A., Hong, H., & Stephenson, C. (2016). Computational Thinking for All: Pedagogical Approaches to Embedding 21st Century Problem Solving in K-12 Classrooms. *TechTrends, 60*, 565–568. https://doi.org/10.1007/s11528-016-0087-7