Factors Influencing Computer Science Learning in Middle School

Shuchi Grover
Center for Technology in Learning
SRI International
Menlo Park, CA 94025
shuchi.grover@sri.com

Roy Pea
Graduate School of Education
Stanford University
Stanford, CA 94305
roypea@stanford.edu

Stephen Cooper
Computer Science Department
Stanford University
Stanford, CA 94305
coopers@stanford.edu

ABSTRACT

In this paper, we describe research conducted around a 7-week curriculum designed to introduce middle school students to computer science with a focus on algorithmic thinking and programming. The pedagogical ideas employed in this curriculum were drawn from past research. Empirical investigations over two studies in a public middle school in the US examined changes in students' understanding of algorithmic constructs and the factors affecting that learning. Multi-level analyses revealed that students in both studies (1) achieved substantial learning gains in algorithmic thinking skills and significant growth towards a more mature understanding of computing as a discipline, and (2) found certain CT ideas and constructs more difficult than others. Prior computing experiences and math and English ability were found to be predictors of learning outcomes. Extracurricular experiences with technology also appeared to impact outcomes.

Keywords

Middle School; Deeper Learning; Learning Factors; K-12 Computer Science Education; Computational Thinking.

1. INTRODUCTION

The rationale for introducing computing in K-12 in order to advance computational thinking (CT) is compelling [10,34,40]. While needs of high school students in the US are being prioritized through courses such as Exploring Computer Science (ECS) and AP CS Principles, there is a growing belief that experiences with computing must start at an earlier age. Middle school years are formative and key for cognitive and social development in the K-12 schooling journey especially with regard to future engagement with STEM fields [35]. Experiences with computing should therefore make middle school amenable to diverse future opportunities as part of students' possible selves.

While there has been some growth in structured middle school computer science (CS) curricula, the development of deeper, transferable CT skills in a classroom setting is yet to be empirically validated. The goal of this research was to address this through designing and empirically studying the use of a structured curriculum for middle school that leverages pedagogical ideas from the learning sciences and computing education research about how children can best develop algorithmic thinking and programming skills. We adopted an iterative process to design, refine, and study our introductory middle school CS course—*Foundations for Advancing Computational Thinking* (FACT) to empirically investigate students' development of CT skills.

This paper focuses on presenting our research around answering the following research questions—**RQ1:** ***What is the variation across learners in learning of algorithmic flow of control (serial execution, looping constructs, and conditional logic) through FACT; RQ2: What factors influence these learning outcomes?*** The paper is organized as follows. A survey of relevant related work presents a backdrop to the research framework and the rationale of FACT's curriculum, pedagogy and assessment design. The research methodology follows including a brief description of the pedagogical designs used to introduce students to computing and algorithmic thinking, and the empirical studies. The Results section provides results in the form of descriptive statistics and also describes the mixed-methods analyses that were used to examine what factors, such as prior experiences and learner backgrounds, explained the variance observed in learning outcomes. The paper ends with a discussion and synthesis of the main findings of this research as well as their implications.

2. RELATED WORK

Much of the CT research involving middle school children in the US has centered on introductory block-based computational tools such as Scratch, Alice, Agentsheets, Blockly, App Inventor etc. Engaging activities such as programming games and apps are used to foster fluency and motivation in computational problem solving. Tangible computational tools such as robotics kits and e-textiles (using Lilypad Arduino) have also been used and studied. Until recently, most of these have occurred in informal settings.

In the formal middle school context, research studies have been conducted as students engage with CT through game design in Alice and Agentsheets [4,28]. Some others have aimed to teach computational thinking in the context of middle school science [e.g. 32,38]. In Denner et al.'s work, students worked through a series of self-paced instructional exercises in Alice and then designed and developed their own games. Students were assessed using a specially designed assessment [37]. They summarize their experiences elsewhere by acknowledging that while computer game programming (CGP) in environments such as Alice have proved to be "a good strategy to attract underrepresented students to computing in middle school and to engage them in programming concepts and systems thinking, CGP does not automatically result in learning, and without some intention on the part of the teacher, it will not result in them learning specific programming concepts." [4]. Previous studies around children and

SIGCSE '16, March 02-05, 2016, Memphis, TN, USA
© 2016 ACM. ISBN 978-1-4503-3685-7/16/03…$15.00
DOI: http://dx.doi.org/10.1145/2839509.2844564

LOGO [19,22,25,26] endorsed the merits of such an approach and underscored that ***teachers as well as instruction play a substantial role in what and how students learn***.

In taking an approach focusing on teaching the foundational concepts of CS through the structured use of Scratch, FACT is similar to efforts in Israel and the UK [31,41]. Our pilot effort involved the design and evaluation of a 7-week "mini-course", to give students a brief introduction to CS and computational problem solving in order to raise interest and awareness in CS and serve as a good foundation for future computing experiences. More importantly, given its duration, FACT can potentially be taught as a unit in existing math, science or technology elective periods. Our work also employs pedagogy for a structured curriculum that makes the teaching of specific CS concepts and vocabulary more intentional, with the goal to consciously ***teach CS***. Since examining students' programming artifacts alone to assess learning can be misleading [19,23], we assess young learners' understanding of concepts of CS through designed assessments in addition to artifacts created by students for evidence of their understanding of computational concepts. This last aspect distinguishes FACT from other efforts in the US [30].

3. METHODOLOGY

This research effort involved two iterations that examined the use of FACT in a public middle school classroom setting. The first iteration was in a traditional face-to-face classroom setting, whereas the second iteration involved investigations on a blended model of learning using an online version of FACT designed and created on Stanford University's OpenEdX MOOC platform. The research involved designing the FACT curriculum; engineering a blended learning experience using an online version of FACT; and empirically investigating FACT's use in a classroom to answer the research question outlined earlier.

3.1 FACT Curriculum

The curriculum (Table 1) was inspired by the 36-week long ECS high school curriculum [8], and includes topics on algorithmic problem solving and programming (in Scratch) that the authors considered foundational and appropriate for middle school students. These map roughly to Units 1, 2 and 4 of ECS. The entire curriculum design effort was guided by goals for "deeper learning" [27], attending to the development of cognitive abilities in addition to interpersonal and intrapersonal abilities (such as collaboration and communication).

3.1.1 Features for Fostering Deeper Learning

To target deeper learning of algorithmic problem solving and to address issues in learning to program that novice programmers face [6,14,25,29], FACT's curriculum design leverages available learning sciences literature. The pedagogy uses a scaffolding and cognitive apprenticeship [2] approach to model the process of programming through the use of (worked) examples [16]. This includes modeling and constructing solutions to computational problems in a manner that reveals the underlying structure of the problem, and the process of composing the solution in pseudo-code or in Scratch. Drawing on past research [9], computing vocabulary and CT language are used during this scaffolding process. FACT also consciously engages with students' narrow perceptions of CS to help them see computing in a new light by using publicly-available videos (bit.ly/CS-rocks) exemplifying computing as a problem-solving discipline with applications in many real-world creative contexts and disciplines [12].

Table 1. 7-week FACT Course for Middle School

Unit 1	Computing is Everywhere! / What is CS?
Unit 2	What are algorithms & programs? Computational solution as a precise sequence of instructions.
Unit 3	Iterative/repetitive flow of control in a program: Loops
Unit 4	Representation of Information (Data and variables)
Unit 5	Boolean Logic & Advanced Looping
Unit 6	Selective flow of control: Conditional thinking
	Final Project (student's own choice; individual or pairs)

FACT emphasizes "learning by doing" [1] for students through a mix of directed projects and meaningful, open-ended projects in Scratch including a substantive culminating project that students were encouraged to do in pairs (Table 2). Frequent low-stakes multiple-choice formative assessments were designed to ensure learners stay engaged with the content and learning goals of the course, and to provide feedback both to the learners and the teacher. These also include questions inspired by Parson's puzzles [24]. FACT aims to foster a deeper understanding of foundational concepts by providing learners opportunities to work with plain English, pseudo-code as well as programs coded in Scratch. At occasional junctures during the course there are opportunities to examine the same algorithm put together in a language besides Scratch. Student learning was also assessed through the final open-ended game design project of choice. FACT's curriculum and assessments are detailed in earlier publications [13,14,15].

Table 2. Sample programming projects in Scratch

Programming Assignments	**Algorithmic ideas**
Share a recipe	Sequence of instructions
(Scratch) Make a life cycle of choice to use in a 6th grade science class	Serial execution
(Scratch) Draw a Spirograph from a polygon of choice	Simple nested loop (also, creative computing (CC))
(Scratch) Create a simple animation	*Forever loop*
(Scratch) Generic polygon maker	Variables; user input
Look inside Scratch code and explain the text version of code	Algorithms in different forms
(Scratch) Draw a 'Squiral'	Loops, variables (& CC)
Open-ended project (in pairs): Create a game using "Repeat Until"	*Loops ending with Boolean condition*
(Scratch) Maze game	Event handlers; Conditionals
(Scratch) Guess my number game	Loops, conditionals, variables, Boolean logic
(Scratch) Final project of choice	*All CT topics taught*

3.2 Study and Data Measures

Empirical studies were conducted in a public middle school classroom in Northern California. Two iterations (Study1 and Study2) of research were conducted with two different cohorts in the same 'Computers' elective class that met four days a week for 55 minute periods. The student samples comprised 7th and 8th grade students (Table 3; ELL="English Language Learners"). In Study1, the course was taught face-to-face by the first author. Study2 was conducted in the same classroom with a new cohort and used a refined version of FACT (based on learning from Study1) on OpenEdX for blended learning and comprised a mix of individual and collaborative activities. The regular classroom teacher, who did not have a background in CS or programming, was present in the classroom at all times assisting with classroom management and also "learning right alongside the students."

Table 3. Student samples in Study1 & Study2

Study	Mean Age	Male	Female	Grade 7	Grade 8	ELL	Special Ed.
1	12.9	21	5	15	11	4	2
2	12.3	20	8	16	12	3	1

3.2.1 Data Measures

The following constituted the main data measures:

- *Prior Experience Survey*: Students were given an extensive survey on prior experiences in technology and computational activities, especially programming.
- *CS Interest & Attitudes Pre-post Survey*: Inspired by [7]
- *Pre-Post Measures of CT***:** Students were given pre-post tests that measured their understanding of computational concepts. The pretest and posttest used items from prior research involving middle school CS with Scratch [7] and included 6 out of 9 questions from the Israel National Exam described in [42]. We did not incorporate the other three questions, as we did not have details on them prior to our study's launch.
- *Course Experience Survey*: to gather feedback from learners for improvements for future iterations.
- *School data:* demographic information including age, gender, and academic placement (English, Math, special needs)

4. ANALYSIS & RESULTS

In order to answer RQ1, pre-post data were first analyzed separately for each study using descriptive statistics, paired *t-tests* and non-parametric tests to study within-subject differences from pre-to-post test. A comparative analysis was also conducted on the six questions of the posttest that were employed in the National CS Exam administered to about 4000 middle school students in Israel in 2012 [42].

Table 4. Pre- and Posttest Scores (out of 100), Study 1 & 2

	Pretest	Posttest		
Study	Mean (SD)	Mean (SD)	t-stat	p-value
1	36.3 (18.2)	79.4 (17.5)	-17.3	<0.001
2	28.1 (21.18)	81.6 (21.0)	-15.5	<0.001

Note: *Non-parametric Mann-Whitney (Wilcoxon) rank sum test was also conducted to test the difference between the pretest and posttest, and the results remained significant.*

Table 5. Posttest Scores by CT Topics, Study1 & Study2

	Study 1	Study 2		
	Mean (SD)	Mean (SD)	t-stat	p
Overall	78.6 (17.1)	81.6 (21.2)	-0.6	0.56
By CS Topic				
Serial Execution	97.4 (13.1)	91.1 (20.7)	1.4	0.18
Conditionals	84.5 (19.0)	84.9 (20.5)	-0.1	0.94
Loops	74.1 (21.9)	77.2 (26.3)	-0.5	0.64
CS Vocabulary	68.2 (17.9)	77.4 (22.2)	-1.7	0.10

(**Note***: t-statistic and p-value refer to the test of difference in the mean scores for Study1 and Study2)*

Relevant learning outcomes pertinent to RQ1 are shown in Tables 4 and 5. On the posttest score, there was no significant difference between the studies. The pretest scores in the two studies were also not statistically different. The pre-to-posttest effect size (Cohen's *d*) on the CT test was roughly 2.4 in both studies. *Learners found serial execution to be easiest and loops, especially those involving variable manipulation, the most difficult to grasp* (Table 5). The comparative analysis with the results from the Israeli nationwide exam revealed comparable or slightly better performances by our students (Figure. 1; Question numbers are the same as in [42]). Some gender differences were observed with girls performing better than boys, however the small number of females in the samples precluded drawing deeper conclusions.

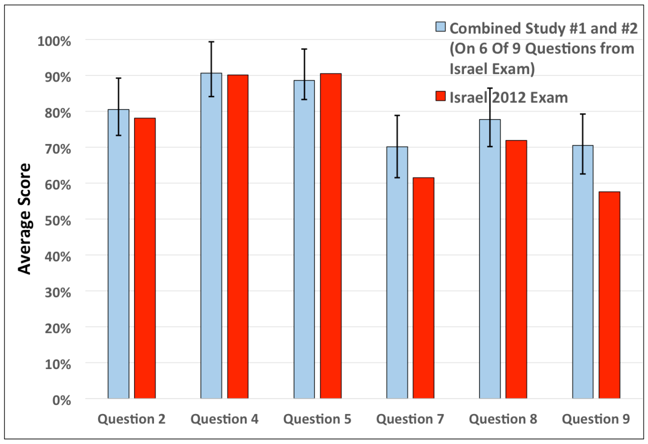

Figure 1: Comparison of Student Performance in Study1 & Study2 vs. 2012 Israel results (on 6 of 9 questions)

To answer RQ2, factor analyses on prior experience variables and multivariate regressions were conducted to determine which factors predicted outcome measures of interest (Section 4.1).

4.1 Factors Influencing Learning Outcomes

What factors explain these differences in learning outcomes? Learning is influenced by several factors, including learners' prior experience, interests and attitudes towards the subject being taught in addition to academic preparation, especially in foundational subjects such as mathematics and English [36]. While it is acknowledged that mathematics and English levels are often a function of learners' SES (socio-economic status), inclusion of SES variables was outside of the scope of this research (as was school context since both studies were conducted in the same school). In order to explain the variability, factor analyses and multivariate regressions analyses were carried out.

4.1.1 Rationale for combined regression analyses

Factor analyses and regressions were conducted on the combined sample from the two studies *to gain more statistical power with a larger sample size (N=54)*. The justification for combining the data was as follows: Since both studies were conducted in the same school and classroom, there were several similarities among the participants and school settings. Students in both studies came in with similar levels of knowledge of computing as tested by the pretest (there was no statistical difference between the studies). The two groups were similar in their mathematics and English abilities as measured by the STAR California state test, and the responses on the survey questions probing interests and attitudes towards CS both before and after the intervention were not statistically different between groups. Students' performances on the posttest were also not statistically different across the two studies (Table 5).

4.1.2 Factor Analysis on Prior Experience Survey

As a first step towards using multivariate regression analyses to explain the effects of the several possible independent variables on the main outcomes of interest, it was necessary to analyze several item-level responses on relevant questions (Table 6) in the prior experience survey for underlying patterns via exploratory factor analytic procedures. Factor analysis is a multivariate statistical approach commonly used for interpreting self-reporting questionnaires [39].

Table 6. Prior Experience Survey Items in Factor Analysis

Q5: (Yes/No) "Have you ever written a computer program?"
Q9: (9 items) "How many times have you ever created the following using some software on the computer?" [List]
Q11: (13 items) "How would you describe your level of experience with the following computer applications/equipment?" [List e.g. Scratch/Alice/Tynker]
Q12: (9 items) "How many times do you use a computer to do each of the following" [List e.g. play online multi-user games]

*Note: **Q9** measured depth of technology experience on a 4-point Likert scale– 1=never to 4=more than six times; **Q11** measured experience level on a 5-point Likert scale from 1="I don't know what it is" to 5="I'm an expert and can teach someone how to use it"; **Q12** measured frequency of computational and media creation on a 8-point Likert scale– 1=Never to 8=Several times a day*

Three main "prior experience factors" were found. We titled them *Coder*, *Media Creator*, and *Online Consumer* as described in Table 7. Pearson's correlation tests between the three factors indicated the correlation coefficient was less than 0.1 suggesting that these were 3 distinct types of students in the study sample. Pearson correlations also suggested that math ability levels (as measured by STAR scores) were significantly negatively correlated only to *Online Consumer*, and positively correlated with *Coder* and *Media Creator*. The implications of these correlational findings are discussed further in Section 5.

Table 7. Descriptions of Prior Experience Factor Variables

Prior Exp (PE) Factor	Description
PE Factor 1: *Coder*	Strong on programming experience. Especially strong components of factor 1 are a positive response on "have you ever programmed before?" and experience with coding languages.
Factor 2: *Media Creator*	Little to no coding experience but strong association with digital media creation activities such as creating digital movies, music, audio.
Factor 3: *Online Consumer*	Little to no experience with computer programming or creating digital media, however strong associations with playing computer & multi-user online games, watching movie videos.

4.1.3 Multivariate Regressions

Based on prior research [36], it was ascertained that the following variables could influence learning outcomes in FACT:

- Prior knowledge of computational constructs.
- Technology fluency and prior experience with computation.
- Interest in and attitudes towards CS.
- Academic preparation (as measured here by mathematics and English levels in STAR California State tests).
- Learning issues (English Language Learner, or ELL).
- Demographics (age and gender).

Some variables were dropped early on as they were not found to be significant predictors of outcomes of interest in either uni-variate or multivariate regressions with any combination of predictor variables. Age was one such variable. The stepwise regressions used to determine the best model thus included:

1. Mean of the CS attitudes survey items.
2. Mean of the 'future interest in CS' survey item calculated for each student.
3. Prior experience factors (the three factors identified above).
4. Most recent Math STAR test score.
5. Most recent English STAR test score.
6. ELL status.
7. Gender.
8. Pretest score (for posttest).

Following stepwise regressions, interest and attitudes variables were dropped due to lack of significance as predictors of outcome variables, as were gender and English STAR score. Table 8 presents the final full regression used to explain the variation in pretest and posttest scores.

Table 8. Final regression for factors influencing outcomes

	Posttest		Pretest	
Variable	**β**	**SE**	**β**	**SE**
Pretest	0.39^{***}	0.11	–	–
Math STAR	0.40^{**}	3.22	0.34^{**}	2.79
ELL	0.08	7.98	0.06	5.61
PE Factor1 (Coder)	0.14	1.79	0.62^{***}	3.10
PE Factor2 (Creator)	-0.01	2.07	0.13	1.64
PE Factor3 (Consumer)	-0.16	1.58	-0.02	1.53
Constant	–	13.82	–	13.39
N	54		54	
Adjusted R^2	0.53		0.49	

$^{*}p < 0.5$. $^{**}p<0.01$. $^{***}p<0.001$

4.1.4 Performance Based on Prior Experience

While prior programming experience as measured by the pretest was found to be relevant for the posttest, the three prior experience factors did not predict performance on the posttest. This suggested that FACT helped ***all students*** regardless of prior experience as measured by the self-report survey. To further examine how the prior experience factors affected learning outcomes, a median split was used to divide students into 'high' and 'low' groups for the 3 factors– "High Coder"/"Low Coder", "High Creator"/"Low Creator", and "High Consumer"/"Low Consumer". As expected, being a High Coder (as opposed to Low Coder) was beneficial for both pretest and posttest scores on the CT test. Since neither Media Creators nor Online Consumer factors were high on prior programming experience, it was also useful to examine how these sub-groups fared in the posttest. In general, learning gains were higher for High Creator than Low Creator students, and being a Low Consumer helped post-scores and learning gains more than being a High Consumer. These differences weren't significant, but were suggestive that perhaps significant relationships may emerge with larger samples.

5. DISCUSSION

Based on the quantitative results and analyses for the two studies, it appears that FACT helped all learners attain substantial gains in learning of basic algorithmic flow of control in computational solutions. Serial execution was the easiest to learn, as expected. *Between conditionals and loops, learners found loops harder to tackle.* Most of the assessment questions concerning loops required manipulation of variables as well, which seemed to be the hardest topic for students to grasp (based on OpenEdX dashboard data on formative quiz performances [11,15] and assigned programming activities). Both these aspects have been known to be particularly difficult for novice programmers [25,33]. Despite our conscious efforts, students struggled with these topics. Extra attention needs to be given as to how introductory courses could be improved to help learners build deeper understanding of variables and the ways in which they work in loops specifically, and in computational solutions, in general. Some of this may also be related to the level of math preparation as discussed below.

5.1 Prior Math and English Preparation

Regression analyses revealed math performance was a positively correlated predictor for CS posttest performance (even when controlling for the pretest). This correlation has been found in past research [21]. It is not entirely surprising given that abstraction is key to computing and a key skill taught in math [5,18]. Given the difficulties students had with loops and variables (which share a strong relationship with abstract and algebraic thinking) in comparison with conditionals, the link to prior math preparation needs to be probed further. This issue is also significant because students' mathematics preparation historically co-varies with socio-economic status and other indicators of diversity.

English Language Learner (ELL) status was a negative predictor for the text-heavy transfer test [discussed in 14,15]. However, ELL students showed high levels of motivation for their open-ended authentic final projects of choice. They performed better on the associated interviews even though their projects were on the lower end of the complexity scale compared to other class projects [15]. We hope to re-examine these aspects of our curriculum as we iterate on it and study it further with more diverse student populations. However, they do suggest that curricula must pay closer attention to diverse levels of math and English preparation and ensure that ***all*** learners succeed in introductory CS.

5.2 Out-of-school Technology Experiences

The curriculum helped all students achieve significant learning gains irrespective of prior experience (as measured by the self-report survey). Not surprisingly, prior programming experience as measured by the pretest was found to positively predict performance on the posttest. Regressing posttest performance on high and low levels of prior experience factors (that resulted from factor analyses on prior experience survey data) revealed that among students who did not have prior programming experience, those with experience in media creation generally did better than those that did not, and those that engaged only in online gaming and video watching (to the exclusion of programming or media creation activities (i.e. the *Online Consumer*s), did worse. Having a high value for the *Online Consumer* factor was also correlated to poor mathematics and English ability (as measured by STAR tests). This perhaps points to other factors such as low SES and a lack of out-of-school experiences that may be considered intellectually enriching. More broadly, these results suggest that the nature of out-of-school technology experiences have a bearing on computational learning. These factors have important implications for curriculum and assessments design, and support the rationale for courses such as ECS that have a strong equity focus as we attempt to ensure ***computing for all***.

6. CONCLUSION & IMPLICATIONS

This research describes two studies that were conducted in a public school. The studies were conducted in an elective class, which meant learners (who were also mostly male) came into the course with generally high interest and motivation. Despite this limited generalizability, the research makes several unique contributions. It is perhaps the first structured online/blended introductory middle school curriculum that has been through rigorous empirical investigation, where computational thinking learning outcomes are measured through pre-post assessments. It is a curriculum that has been shown to result in learning gains (in the context in which it was studied) and to foster CT skills in middle school students as measured by assessment items used in prior research including a national CS exam in Israel. It serves as an example of course designed on an online platform that effectively employs an iterative research methodology to refine a curriculum aiming to foster learning in a blended classroom setting. The conscious attention devoted to teaching (and assessing) for transfer, as well as attending to perceptions of computing as a discipline through an engaging corpus of publicly available videos (presented in detail in prior publications [12, 15]), are also unique aspects of this research. Breaking down pre-posttest by algorithmic constructs to examine learning on these different aspects highlights targets of difficulty in middle school students. This research thus makes contributions to the design of curricula and assessments using block-based environments such as Scratch, Blockly, and Alice that are popularly used today.

The finding that students with low prior mathematics achievement experienced difficulties in learning CS in middle school has broad importance- it points to a need to build abstraction skills that math prepares students for. Other recent research also points to the relevance of English and math skills in environments such as Scratch for younger grades [17]. Future directions for this effort involve iterating on the curriculum to attend more closely to prior math preparation and to make the curriculum more accessible to all students by continuing to empirically examine its use with broader audiences of middle school students and teachers.

7. ACKNOWLEDGMENTS

We gratefully acknowledge grant support from the National Science Foundation (NSF #1343227 and NSF #0835854).

8. REFERENCES

[1] Barron B., & Daring-Hammond, L. 2008. How can we teach for meaningful learning? In Darling-Hammond, L.,et al. *Powerful learning: What we know about teaching for understanding*. San Francisco: Jossey-Bass.

[2] Brown, J. S., Collins, A., & Newman, S. E. 1989. Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. *Knowing, learning, and instruction: Essays in honor of Robert Glaser*, *487*.

[3] Campe, S., Denner, J., & Werner, L. 2013. Intentional computing: Getting the results you want from game programming classes. *Journal of Computing Teachers*.

[4] Denner, J., Werner, L., & Ortiz, E. 2012. Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts?. *Computers & Education, 58*(1), 240-249.

[5] Devlin, K. 2003. Why universities require computer science students to take math. *Communications of the ACM*, *46*(9).

[6] du Boulay, B. 1986. Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1).

[7] Ericson, B., & McKlin, T. 2012. Effective and sustainable computing summer camps. In Proceedings of the 43rd ACM technical symposium on Computer Science

[8] Goode, J., Chapman, G., Margolis, J., Landa, J., Ullah, T., Watkins, D., & Stephenson, C. 2013. Exploring Computer Science. http://www.exploringcs.org/curriculum

[9] Grover, S. 2011. Robotics and engineering for middle and high school students to develop computational thinking, in *Annual Meeting of the AERA*, New Orleans, LA.

[10] Grover, S., and Pea, R. 2013. Computational Thinking in K–12: A Review of the State of the Field. Educational Researcher, 42(1), 38-43.

[11] Grover, S., Pea, R., & Cooper, S. 2014. Promoting active learning & leveraging dashboards for curriculum assessment in an OpenEdX introductory CS course for middle school. In *Proceedings of the first ACM conference on Learning@ scale conference* (pp. 205-206). ACM.

[12] Grover, S., Pea, R. & Cooper, S. 2014. Remedying Misperceptions of Computer Science among Middle School Students. *In Proceedings of the 45th ACM Technical Symposium on Computer Science Education.* ACM.

[13] Grover, S., Cooper, S., & Pea, R. 2014. Assessing computational learning in K-12. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 57-62). ACM.

[14] Grover, S., Pea, R. and Cooper, S. 2014. Expansive Framing and Preparation for Future Learning in Middle-School Computer Science. In Proceedings of the 11th ICLS (2014).

[15] Grover, S., Pea, R., Cooper, S. 2015. Designing for Deeper Learning in a Blended Computer Science Course for Middle School Students. *Computer Sc. Education*, 25(2), 199-237.

[16] Guzdial, M. 2009. How we teach Introductory Computer Science is wrong. Blog at *Communications of the ACM*.

[17] Hill, C., Dwyer, H. A., Martinez, T., Harlow, D., & Franklin, D. 2015. Floors and Flexibility: Designing a programming environment for 4th-6th grade classrooms. In Proceedings of the 46th ACM Technical Symposium on Computer Science Education (pp. 546-551). ACM.

[18] Kramer, J. 2007. Is abstraction the key to computing?. *Communications of the ACM*, *50*(4), 36-42.

[19] Kurland, D. M., & Pea, R. D. 1985. Children's mental models of recursive LOGO programs. *Journal of Educational Computing Research, 1*(2), 235-243.

[20] Lahtinen, E., Ala-Mutka, K., & Järvinen, H. M. 2005. A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin, 37*(3), 14-18.

[21] Lewis, C. M., & Shah, N. 2012. Building upon and enriching grade four mathematics standards with programming curriculum. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education.* ACM.

[22] Mayer, R.E. 1989. The psychology of how novices learn computer programming. In E. Soloway & J.C. Spohrer (Eds.), Studying the novice programmer (pp. 129–159). Hillsdale, NJ: Lawrence Erlbaum

[23] Meerbaum-Salant, O., Armoni, M., and Ben-Ari, M., 2010. Learning computer science concepts with Scratch. In Proceedings of the Sixth International Workshop on Computing Education Research (ICER '10). ACM.

[24] Parsons, D., & Haden, P. 2006. Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52* (pp. 157-163). Australian Computer Society, Inc.

[25] Pea, R. D., & Kurland, D. M. 1983. On the cognitive prerequisites of learning computer programming. (Tech. Report No. 16). New York: Bank Street College of Education, Center for Children and Technology.

[26] Pea, R. D., & Kurland, D. M. 1984. On the cognitive effects of learning computer programming. New Ideas In Psychology, 2, 137–168.

[27] Pellegrino, J. W., & Hilton, M. L. (Eds.). 2013. *Education for life and work: Developing transferable knowledge and skills in the 21st century*. National Academies Press.

[28] Repenning, A., Webb, D., & Ioannidou, A. 2010. Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE '10)*, 265–269. New York, NY: ACM.

[29] Robins, A., Rountree, J., & Rountree, N. 2003. Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137-172.

[30] Schofield, E., Erlinger, M., & Dodds, Z. 2014. MyCS: CS for middle-years students and their teachers. In *Proceedings of the 45th ACM technical symposium on Computer science education* (pp. 337-342). ACM.

[31] Scott, J. 2013. The royal society of Edinburgh/British computer society computer science exemplification project. *Proceedings of ITiCSE'13*, 313-315.

[32] Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G., & Clark, D. 2013. Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies*, 1–30.

[33] Spohrer, J. C. & Soloway, E. 1986. Novice mistakes: are the folk wisdoms correct? *Communications of the ACM*, 29(7.

[34] Stephenson, C., Gal-Ezer, J., Haberman, B., & Verno, A. 2005. The new educational imperative: Improving high school computer science education. *Computer Science Teachers Association (CSTA), New York, New York.*

[35] Tai, R., Qi Liu, C., Maltese, A.V., Fan, X. 2006. Planning Early for Careers in Science. Science. 312(5777) 1143-1144

[36] Wang, M. C., Haertel, G. D., & Walberg, H. J. (1993). Toward a knowledge base for school learning. *Review of educational research*, *63*(3), 249-294.

[37] Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. 2012. The Fairy Performance Assessment: Measuring Computational Thinking in Middle School. In Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE '12), 215-220. ACM, New York, NY.

[38] Wilensky, U., Brady, C., & Horn, M. 2014. Fostering Computational Literacy in Science Classrooms. *Communications of the ACM. 57*(8): pp 17-21.

[39] Williams, B., Brown, T., & Onsman, A. 2010. Exploratory factor analysis: A five-step guide for novices. *Australasian Journal of Paramedicine, 8*(3). Retrieved from http://ro.ecu.edu.au/jephc/vol8/iss3/1

[40] Wing, J. 2006. Computational Thinking. Communications of the ACM. 49(3), 33-36.

[41] Zur Bargury, I. 2012. A new Curriculum for Junior-High in Computer Science. *ITiCSE'12,* pp. 204-208. ACM.

[42] Zur-Bargury, I., Pârv, B., & Lanzberg, D. 2013. A nationwide exam as a tool for improving a new curriculum. In Proceedings of ITiCSE'13 (pp. 267-272). ACM.