From Scratch to "Real" Programming

MICHAL ARMONI, ORNI MEERBAUM-SALANT, and MORDECHAI BEN-ARI,
Weizmann Institute of Science

Computer science (CS) activities for young students are widely used, particularly visual programming environments. We investigated the use of the Scratch environment for teaching CS concepts to middle school students. In a previous article [Meerbaum-Salant et al. 2013], we reported on the extent to which the CS concepts were successfully learned. In this article, we look at the transition from studying CS with the visual Scratch environment in middle school to studying CS with a professional textual programming language (C# or Java) in secondary school. We found that the programming knowledge and experience of students who had learned Scratch greatly facilitated learning the more advanced material in secondary school: less time was needed to learn new topics, there were fewer learning difficulties, and they achieved higher cognitive levels of understanding of most concepts (although at the end of the teaching process, there were no significant differences in achievements compared to students who had not studied Scratch). Furthermore, there was increased enrollment in CS classes, and students were observed to display higher levels of motivation and self-efficacy. This research justifies teaching CS in general and visual programming in particular in middle schools.

Categories and Subject Descriptors: K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer science education*

General Terms: Human Factors

Additional Key Words and Phrases: Scratch, computer programming, middle schools, spiral curriculum

ACM Reference Format:
Michal Armoni, Orni Meerbaum-Salant, and Mordechai Ben-Ari. 2015. From Scratch to "real" programming. ACM Trans. Comput. Educ. 14, 4, Article 25 (February 2015), 15 pages.
DOI: http://dx.doi.org/10.1145/2677087

1. INTRODUCTION

Why teach computer science (CS) to middle school children? A common answer is that early exposure to CS will combat stereotypes: CS is only for asocial nerds [Carter 2006] and CS is only for boys [Margolis and Fisher 2003]. Successful participation in CS activities at an early age should be able to improve students' self-efficacy and therefore increase their motivation to attempt CS studies at the secondary and even tertiary levels.

We are interested in a separate but related issue: to what extent can middle school students learn CS, and how does this affect their encounter with CS in secondary school? If teaching CS in middle school only led to higher levels of motivation and self-efficacy, that in itself would justify investing the time and resources. However, as

This work was partially supported by the Israel Science Foundation grant 09/1277 and by a Sir Charles Clore Postdoctoral Fellowship.
Authors' addresses: M. Armoni, O. Meerbaum-Salant, and M. Ben-Ari, Department of Science Teaching, Weizmann Institute of Science, Israel; emails: {michal.armoni, orni.meerbaum-salant, moti.ben-ari}@weizmann.ac.il.

DOI: http://dx.doi.org/10.1145/2677087

educators, we are also interested in whether learning occurs or not. Furthermore, we feel that successful learning is not a totally separate issue, because it is hard to see how self-efficacy and motivation will be increased if no actual learning takes place.

This article presents the second half of a long-term research project on teaching CS in middle schools. The first part investigated the learning of CS concepts using the visual programming environment Scratch [Meerbaum-Salant et al. 2013]. We discovered that certain concepts were learned more successfully than others, and we were able to suggest how to improve learning by better teacher training and by the development of appropriate learning materials. The second part of the research project focused on the transition from learning CS in middle school with Scratch—an environment designed specifically for children—to learning CS in secondary school using a "real" programming language and a professional software development environment. If we could show that learning CS at the secondary school level was facilitated by learning CS in middle school, it would amplify the justification for the latter based primarily on affective considerations.

2. BACKGROUND

2.1. CS for Children

Many types of activities have been proposed for introducing CS to children. They fall into three categories:

—*Kinesthetic activities*: These are activities that do not use a computer, such as *Computer Science Unplugged* [Bell et al. 2005] and magic shows [Curzon and McOwan 2008]. In a previous research project, we could not find improvement in attitudes toward CS after participation in CS Unplugged activities [Taub et al. 2012]. We suggested that such activities need to be more than just fun; they need to be explicitly connected to CS in the real world, and we proposed ways of improving these activities. *Abenteur Informatik* [Gallenbacher 2008] is a set of kinesthetic activities (in German) that explicitly makes these connections and is therefore likely to be successful.
—*Visual programming environments* (see the survey by Kelleher and Pausch [2005]): These environments enable young people to construct running programs by greatly simplifying the interface (iconic rather than textual) and by increasing interest (constructing animations instead of computations). Two visual programming environments, Scratch [Resnick et al. 2009] and Alice [Dann et al. 2009], have enjoyed phenomenal success.
—*Robotics*: Robotics activities are widely used to introduce science and technology to young students [Druin and Hendler 2000]. Robotics are appropriate for introducing CS (and other subjects such as physics and mathematics) to young people because the algorithms and programs are reified in concrete objects and not just as virtual characters on screen, as in Scratch and Alice.

In this article, we describe our research with the Scratch visual programming environment. We believe that visual programming environments enable students to engage at a deep personal level with central CS concepts in algorithms and programming. The advantage of visual programming environments can be concisely expressed using Seymour Papert's term *hard fun*: "I have no doubt that this kid called the work fun because it was hard rather than in spite of being hard" [Papert 2002]. We chose Scratch because it supports advanced CS concepts and yet its two-dimensional world is quite simple when compared with the three-dimensional world of Alice.

2.2. Scratch

Scratch (http://scratch.mit.edu), created by the Lifelong Kindergarten Group at the MIT Media Laboratory, is a media-rich software development environment for novices [Resnick et al. 2009]. Programs in Scratch are composed of scripts that are built by dragging and dropping blocks that represent program components, such as expressions, conditions, statements, and variables. This mode of interaction prevents the trivial syntax errors that can cause students to become frustrated. Scratch programs implement animations of graphical sprites that are displayed on the computer's screen. In the current version, Scratch 2.0, students can use the environment directly from a Web browser. Our research used the previous version (Scratch 1.4), which requires that the software be downloaded and installed; this did not cause any problems.

The literature survey in Meerbaum-Salant et al. [2013] describes previous work on the use of Scratch both in formal and informal educational settings. The only work directly related to this article involves the use of Scratch to introduce CS at the *tertiary level*. Malan and Leitner [2007] of Harvard University taught Scratch in the first week of a broad introduction to CS before teaching Java. In a survey, most of the students—essentially all students with no programming experience—felt that learning Scratch was a positive experience. As expressed by one student:

> Though we did not learn Java syntax by using Scratch, we learned the type of thinking necessary to implement simple programs. . . . I was able to approach the first Java programs with an idea of how to tackle the problems. Though I did not yet know how to create a for-loop, I knew when a for-loop was necessary because I had used loops in my Scratch program [pp. 226–227].

Similarly, Wolz et al. [2009] used Scratch for 3 weeks of an introductory course at the College of New Jersey and reported that Scratch helped the students:

> Although the results are purely anecdotal, students transfer these problem solving skills to mainstream languages [p. 3].

Dann et al. [2012] developed a version of Alice that works as a plugin to NetBeans to facilitate a smooth transfer of experience in Alice to Java through *mediated transfer*. Students first studied Alice for 2 weeks and then Java was introduced using two techniques that characterize mediated transfer: *bridging*, where teachers help students build bridges from one context to another, and *hugging*, where teachers construct new learning situations to be similar to previous ones. Students studying in the experimental group with mediated transfer scored a full letter grade higher than those in the control group from a previous semester who did not use mediated transfer.

Our research is different because it investigated the transition at a much earlier educational level and over a much longer period of time. Furthermore, its methodology goes far beyond the simple surveys used by Malan and Leitner [2007] and by Wolz et al. [2009].

2.3. Spiral Curriculum

This research has its foundation in the concept of a spiral curriculum [Bruner 1960], where the learning of a topic is repeated, each time in greater depth building on the previous cycle. Scratch seems particularly appropriate for a spiral curriculum because it supports the intuition, intrinsic interest, and discovery learning that Bruner emphasized. Bruner [1966] proposed that learning occurs in three stages:

—*Enactive*: Manipulating concrete objects
—*Iconic*: Manipulating images of objects
—*Symbolic*: Manipulating representations of objects.

Table I. The Research Setting

Group	School	Teacher	Students	Number Who Had Learned Scratch
C1	S1	T1	16	12
C2	S2	T2	25	15
C3	S2	T2	20	17
C4	S3	T3	20	0
C5	S4	T4	39	0

Clearly, Scratch enables iconic learning and possibly also enactive learning if we consider clicking on buttons and moving sliders to be the manipulation of concrete objects. Scratch also enables symbolic learning, because objects like the count of aliens hit are manipulated in their representation as variables. However, it is only when learning to program with "real" programming languages that learning must become almost totally representational. The emphasis on an iconic style of programming in Scratch should facilitate the transition to the more abstract representational style of "real" programming languages.

3. THE RESEARCH

3.1. Research Question

We posed the following research question:

> Does learning Scratch in middle schools lead to improved learning of CS at the secondary level? If so, how does it contribute to improved learning?

We conjectured that learning Scratch would facilitate learning CS at the secondary level, both in terms of cognitive aspects and in terms of affective aspects.

3.2. Research Context

The research took place during the 2011–12 school year. All students who participated in the study were 10th-grade students (15 to 16 years old) taking the course Foundations of Computer Science [Armoni et al. 2010], which is the first course in the Israeli high school CS curriculum [Gal-Ezer et al. 1995]. Within that curriculum, it is a mandatory course that introduces students to concepts and ideas of CS, interleaved with programming practice (in Java or in C#, according to each teacher's preference). The population included 120 students, studying in five classes, in four schools, and taught by four teachers. Two of the classes consisted of students who had not previously taken a CS course in school. Three of the classes were mixed: some students had not previously taken a course in CS, whereas other students had studied a course on CS concepts with Scratch during the previous year (9th grade). These students had participated in our first study [Meerbaum-Salant et al. 2013]. In total, 44 students (the experimental group) had taken the 9th-grade course, whereas 76 students (the control group) had not. The research population is shown in Table I.

To perform research in an authentic setting—actual classes learning for several months—we had to accept the assignment of students to classes that the secondary schools made according to their own criteria. Nevertheless, the mixed classes were probably more of an advantage than a disadvantage, because it allowed us to observe the mutual influence of students who had learned Scratch and those who had not.

3.3. Methodology

In the previous study, we looked at many concepts that were both fundamental in CS and central to developing programs in Scratch: initialization, repeated execution (unbounded, bounded, conditional), conditional execution, communication by message

passing, variables, event handling, and concurrency. For this study, we could only look at those concepts that were in the intersection of these Scratch concepts and the concepts taught in the Foundations course. The Foundations course includes topics such as arrays and objects, which are not supported in Scratch, whereas it does not include concepts related to concurrency and events. Therefore, the study is restricted to the first part of the Foundations course (from beginning of the school year in September until the end of January), during which the students are taught the following concepts: variables, conditional execution, bounded repeated execution, and conditional repeated execution.

This was mixed-method research where we combined quantitative and qualitative approaches [Johnson and Onwuegbuzie 2004]. The research tools included multiple tests from which we could obtain quantitative results on the learning of concepts, together with observations and interviews from which we could discover how the students and teachers reacted to the situation of learning CS after the course on Scratch.

3.3.1. Tests. Eight tests were given:

—*Tests of individual concepts*: Six tests focused on individual concepts and were given close to the completion of the learning process of the tested concept. The questions on the tests were designed to check the learning at several cognitive levels as explained later. There was one test on the concept of variables, two on conditional execution, two on bounded repeated execution, and one on conditional repeated execution.
—*Interim test*: This test measured the students' combined achievements on three of the four concepts (not including conditional repeated execution). This test was still close to the completion of the teaching process of these concepts, but it enabled us to examine the achievements of students when required to work with several concepts at a high cognitive level.
—*Final test*: A final test was given a few weeks after the test on conditional repeated execution. It was a relatively long test with complex questions. It enabled us to examine the results of the teaching process after time had passed and from a holistic point of view rather than looking at each concept separately.

3.3.2. Observations and Interviews. Following the cognitive anthropology tradition [Jacob 1998], the second author was a participant passive observer and took extensive field notes in one of the experimental (mixed) classes, C2, with occasional observations in C3. The observations in C2 were conducted for 6 hours per week over 5 months, whereas C3 was observed on three occasions. During the observations, everything that happened in class was documented in the researcher's diary to allow for latter analysis in the spirit of grounded theory [Glaser and Strauss 1975]. In other words, the documentation was not restricted or filtered by predefined categories. Since this kind of observations necessitate a longitudinal practice, going back to the same group of students over and over again, only one class, C2, was observed. C3 was only observed occasionally to ensure that the experience of teaching and learning in the observed class was not exceptional. As noted later, this was also supported by the quantitative findings.

Again, in line with the cognitive anthropology tradition, semi-open informal interviews were conducted by the second author after the final test with 10 students who were randomly chosen by the teacher from C2 and C3. They were asked about their experience in the course, their impressions, and their attitudes toward CS. These were high-level guiding questions. Depending on students' answers, the interviewer used informal follow-up questions to clarify their answers, show interest in their answers, or encourage them to share their feelings and talk freely. The informal protocol was chosen to create a friendly setting and a comfortable atmosphere. As was the case

regarding the observations, the rich data collected through these informal interviews allowed for an analysis in the spirit of grounded theory.

To reinforce and clarify the results of the observations and interviews with the students, several interviews were conducted with teacher T2 who taught C2 and C3, since the observations were conducted in C2 (and occasionally in C3). The interviews with T2 were short reflective interviews; in addition, a final interview asked for her views and impressions on the teaching and learning processes, as documented during the observations. Again, the purpose of these interviews was to clarify events that had occurred in class and were recorded during the observations, and to triangulate the observer's impressions with the teacher's point of view.

The other teachers were interviewed occasionally to monitor the teaching process in their classes—that is, to follow what concepts were taught up to a certain point, for how many hours, and so forth.

As noted earlier, the rich data from the observations and interviews were analyzed following the tradition of grounded theory. The class field notes and the transcripts of the interviews were read, allowing for patterns to emerge from the data. This process involved fragments of text being ascribed to existing patterns or inducing the definition of new ones. Once patterns (e.g., familiarity with concepts or restriction of the recognition, presented later) were identified and discussed among the research team to reach a stable set, a rescanning of the data took place using the set of emerging patterns as categories for analysis or as prisms to look through at the data. This analysis was conducted by the second author and validated by the other two authors, who independently ascribed fragments of text to the different categories. Very few cases of disagreement occurred, and these were resolved by discussion.

3.3.3. The Taxonomy. The design of the questions included in the tests and their analysis were guided by the taxonomy that we developed during the first part of the research [Meerbaum-Salant et al. 2013]. This is a two-dimensional taxonomy that combines the SOLO taxonomy [Biggs and Collis 1982] and the revised Bloom taxonomy [Anderson et al. 2001]. The first dimension consists of three levels of the SOLO taxonomies [Meerbaum-Salant et al. 2013]:

—*Unistructural*: A local perspective where mainly one item or aspect is used or emphasized. Others are missing and no significant connections are made.
—*Multistructural*: A multipoint perspective, where several relevant items or aspects are used or acknowledged but significant connections are missing and a whole picture is not yet formed.
—*Relational*: A holistic perspective in which meta-connections are grasped. The significance of parts with respect to the whole is demonstrated and appreciated.

The second dimension consists of three levels of the Bloom taxonomy adapted to CS [Meerbaum-Salant et al. 2013]:

—*Understanding*: The ability to summarize, explain, exemplify, classify, and compare CS concepts, including programming constructs.
—*Applying*: The ability to execute programs or algorithms, to track them, and to recognize their goals.
—*Creating*: The ability to plan and produce programs or algorithms.

The result is a hierarchy of categories corresponding to cognitive levels from the lowest level of Unistructural Understanding to the highest level of Relational Creating.

A question corresponds to a certain level if a correct and complete solution requires working at that level. Since we were limited in the length of the tests that could be administered during a single class session, we could not cover all categories of the taxonomy in each test. We preferred to ask questions that assessed higher cognitive

Table II. Measurement of Concepts by Cognitive Levels

Variables	Conditional Execution	Bounded Repeated Execution	Conditional Repeated Execution
Multistructural Applying	Multistructural Applying	Multistructural Applying	Multistructural Applying
Multistructural Creating	Multistructural Creating	Relational Understanding	Multistructural Creating
	Relational Applying	Relational Applying	Relational Understanding
	Relational Creating	Relational Creating	

levels. Table II describes the different cognitive levels for which each of the four concepts was examined. Examples of the questions on different cognitive levels can be found in the Appendix.

The construction of a question according to a cognitive level was performed by the second author. The other two authors independently assigned cognitive levels to a sample of the questions. In the event of disagreement, the majority opinion was used.

The interim test measured students' combined achievements on three of the concepts using questions categorized in the highest cognitive level of Relational Creating. The questions on the final test were complex, covering all of the concepts. Each of the questions in the interim and final tests was categorized only by the cognitive level required for its solution rather than by a combination of a concept and a cognitive level. Content validity of all questions in all tests (i.e., ensuring that each of the questions indeed examined learning of the intended concept) was established by a process of expert agreement [Delgado-Rico et al. 2012], where the other two authors independently verified that each question examined learning of the intended concept or combination of concepts.

We took a student's grade on a specific question to reflect his or her performance at the cognitive level of the question. The questions were graded in the range 0 to 100, and partial credit was given where appropriate. The grading was performed by the second researcher, who constructed a grading rubric. The grading focused on semantics, and grades were not reduced if there were syntactical errors. A sample of the grades was checked by the other researchers for reliability.

Since the teaching process in class C5 (a control group class) was a bit different and also somewhat slower than in the other classes, by the end of January the teacher had not taught conditional repeated execution. The data collection in this class was partial, and the class did not participate in the last three tests.

4. FINDINGS

In this section, we describe our findings. Section 4.1 presents the findings regarding the learning process and learning outcomes, and Section 4.2 presents findings on affective aspects related to the learning process.

4.1. The Learning Process and Outcomes

We first present the quantitative findings from the analysis of students' solutions to the tests and then the qualitative findings from the analysis of the observations and interviews.

4.1.1. Quantitative Findings. Table III presents the average grades for the different concepts at the different cognitive levels for the experimental and control groups. These grades are from the six concept-specific tests and do not include the interim and final tests, which are discussed later. The grades were aggregated from all questions that were associated with a specific concept and a specific cognitive level. The number of

Table III. Students' Grades by Concepts and Cognitive Levels

	Multistructural Applying		Multistructural Creating		Relational Understanding		Relational Applying		Relational Creating	
	EXP	CTR	EXP	CTR	EXP	CTR	EXP	CTR	EXP	CTR
Variables	80 (24)	80 (63)	74 (24)	69 (63)	—	—	—	—	—	—
Conditional Dxecution	60 (31)	62 (59)	65 (42)	72 (68)	—	—	27 (42)	36 (63)	86 (42)	83 (68)
Bounded Repeated Execution	89** (38)	65** (67)	—	—	64* (38)	48* (67)	89* (38)	73* (67)	85** (31)	46** (29)
Conditional Repeated Execution	80** (33)	61** (29)	96** (33)	49** (29)	80** (33)	47** (29)	—	—	—	—

Significant differences: *, $p < .04$; **, $p < .004$.

Table IV. Students' Grades by Cognitive Levels (Final Test)

	Multistructural Applying	Relational Understanding	Relational Creating
Experimental Group	85 (30)	30 (30)	51 (30)
Control Group	71 (21)	14 (21)	30 (21)
p	Not significant	Not significant	0.0427

students who answered one or more questions associated with each concept-level pair is given in parentheses.

There were no significant differences between the experimental and control groups for the concepts of variables and conditional execution. For repeated execution (bounded and conditional), significant differences were found at all cognitive levels, in favor of the experimental group.

The tests were given immediately after teaching each concept, but it seems that most of these differences did not last. In the interim test that measured the combined achievement on three of the concepts (variables, conditional execution, bounded repeated execution) at the highest cognitive level of Relational Creating, there were no differences between the experimental and control groups, both of which obtained average scores of 88.

The final test also measured the outcomes of the learning process on a more holistic level rather than for each concept separately. Table IV presents the scores for this test for the cognitive levels Multistructural Applying, Relational Understanding, and Relational Creating. The experimental group did better, but only at the highest level of Relational Creating was the difference significant.

4.1.2. Qualitative Findings. The qualitative findings are based on the observations in C2 (with occasional observations in C3) and interviews with students, triangulated by reflective interviews with T2, the teacher of the observed classes. As noted earlier, the longitudinal deep nature of the observations allowed us to focus on one class only. However, no significant differences were found between the mixed classes during the quantitative analysis, thus we could safely assume that the teaching and learning process that took place in the observed class was not exceptional, as was supported also by the occasional observations in class C3.

We describe the findings for each of the patterns that emerged during the grounded analysis of the qualitative data.

Familiarity with concepts. We observed that students from the experimental group recognized concepts early in the teaching process. This sometimes happened even

before the teacher had a chance to explain anything. For example, in C2, the teacher T2 wrote a code fragment on the board that used variables, but before she began to explain this code, students in the experimental group shouted that they recognized the new entity as variables.

A similar event was also observed in class during the teaching of conditional execution. The teacher reflected on this event during an interview:

> *T2*: The moment I introduced conditional execution to the students, the Scratch students made the necessary connections to the material that they had learned through Scratch.

When repeated execution was introduced in class, students from the experimental group seemed excited; one of them immediately said, "It is the same as in Scratch." This was reinforced by the teacher who said, "They immediately knew what a loop is."

In summary, during the observations, the pattern of familiarity with concepts was evident, as summarized nicely in the words of the teacher:

> *T2*: When I explained the new material the [Scratch] students understood the concept and made the right connections with their former material.

Restriction of the recognition. Although students recognized concepts, they often did so only in the form that they had encountered them in Scratch. For example, during the previous episode where the Scratch students recognized variables, they explained variables as a means for counting points in a game.

A further difficulty occurred when concepts were differently implemented in the languages. Scratch is a dynamically typed language, whereas Java and C# are statically typed (except for polymorphic types, which are not studied at this level). Typed variable declarations presented somewhat of a challenge for the Scratch students. One of them made the following remark:

> *S1*: What is the word before the variable?. . . I really don't understand why we have to define the variable type and I don't understand the necessity of the type casting.

Restriction also occurred in the context of conditional execution. In our previous study [Meerbaum-Salant et al. 2011], we saw that students used Scratch control structures in a restricted manner; specifically, they rarely used conditional execution with an alternative (if-else). This restriction affected their subsequent learning:

> *S2*: My solution is different. I only used the if-statements. I didn't use the else-statements and it's worked for me on the computer.

Scratch encourages the use of unbounded loops (forever and forever-if). Some students developed a concept of repeated execution that was restricted to this case. However, in C# and Java, repeated execution is almost always bounded (for- and while-loops). When loops were first taught in C#, one student immediately said, "Loops are forever."

The Scratch students rapidly learned to use typed variables and different control structures, showing that the restricted interpretations were not persistent and were quickly overcome.

Shortened teaching process. The observer, who is also an experienced CS teacher who had taught the Foundations course for several years, had the impression of a shortened teaching process regarding the concepts investigated in this study. These impressions were supported by the views of the two teachers of the mixed classes, who reported that knowledge of Scratch enabled them to shorten the amount of time devoted to each concept. These views are demonstrated here by excerpts from the interview with T2:

T2: There are many major differences in the teaching process between the students who learned Scratch compared with the students who don't have this background. The Scratch students understand the material faster.

T2: I felt I could teach the material faster, compared to previous years.

T2: Exposure to Scratch helped me a lot this year. In all the years I taught, it took me a number of lessons to teach each subject. This year it took me less time. The teaching process becomes more streamlined and it shortened the duration of the teaching process.

This can be specifically exemplified for the relatively complex concept of variables as counters (a concept that encompasses the algorithmic pattern of counting) [Muller 2005]. Since the Scratch students were familiar with variables, they actually identified them with counters, as mentioned earlier, as a means for counting points in a game. Thus, not surprisingly, we observed that the task of explaining the need for counters in class was relatively easy. This impression was shared by the teacher:

T2: I have been teaching for many years. Usually, when I am teaching [the concept of] counter it takes me a long time to explain to the students the necessity of the counters.

The observer had a similar impression regarding the concept of conditional execution. Again, this impression was shared and reinforced quantitatively by the teacher:

T2: In the past, it took me six lessons to teach conditions [conditional execution]. Now it took me only three lessons, half of the time.

Reduced difficulties in teaching and learning. The improvement was not only in terms of the time needed to teach each concept but also in terms of understanding the concepts.

In the interviews, it has indeed emerged that students were aware that experience with Scratch streamlined the learning process:

S3: It [C# and Scratch] is [based] on the same principle, when I think of it. . . . It helps me to understand better this year. As if, people come, and they were not in [the] Scratch [class], no, they don't know at all what they enter into. I, for me it helped understanding the. . . the beginning, how, how am I supposed to construct, how do I start. It helped me understand.

Even in the case where Scratch students had a restricted view of a construct (e.g., restricting repeated execution to unbounded loops), they were able to quickly extend their knowledge:

S3: In Scratch you have forever if and forever. So this, immediately. . . we have the loops [the student is referring to loop constructs in C#] that are [based] on the same principle. It [the repeated execution constructs learned in Scratch] is something that has certainly helped me to understand the whole point of loops.

This was supported by the impressions of the teacher:

T2: I used to have difficulties in [students'] understanding of the purposes of the conditions [conditional execution] and now I only deal with syntax errors, such as missing brackets, instead of logical errors.

T2: This year I had to deal with syntax errors instead of problems in understanding the purpose.

Consistent with the quantitative results, the teachers felt that there was no noticeable difference in understanding at the end of the year but that the previous experience contributed to the teaching process:

> *T2*: There were no major differences in the students' success at the end of the year. The differences were expressed in the students' comprehension of the concepts [when the concepts were initially taught]. When I taught the basic concepts there were gaps in the knowledge [between the students who studied Scratch and those who didn't]. The Scratch students knew the basic concepts.... This exposure helped me a lot in the teaching process. ... The [Scratch] students had images and understood the purpose of each concept.

One difficulty that was observed and also reported by T2 was that Scratch students struggled initially with the use of English keywords in C# and Java. The Scratch environment supports many natural languages, and the students had found it easy to work with their native language.

4.2. Affective Aspects

4.2.1. Motivation and Engagement. In our previous article, we discussed motivation as it emerged from interviews with students [Meerbaum-Salant et al. 2013]. Our current findings are consistent with what we presented there.

Many of the Scratch students chose to study CS in high school. After teaching Scratch in middle school, one secondary school opened two CS classes (C2 and C3), whereas only one class had been offered in previous years. In the other secondary school, the number of CS students rose from 8 to 28 (class C1) after Scratch was taught in middle school.

Indeed, the students reported that Scratch changed their perception on CS and affected their decision to choose CS in secondary school:

> *S4*: The lessons [of Scratch] were interesting and this is why I decided to choose CS.

> *S5*: [I chose CS] since Scratch interested me and I wanted to continue in this area.

> *S6*: It [Scratch] helped me become interested.

> *S7*: Knowing Scratch helped me in that I enjoyed constructing games, and in choosing CS.

In the mixed classes, this even had some positive effect on the other students who were not exposed to Scratch:

> *S8*: Because I didn't learn Scratch last year, I started to learn in the summer vacation the material of the next year. I was afraid that I would not be prepared. I wanted to fill in the gap so that I can also succeed.

The teachers also remarked on the high motivation of the Scratch students:

> *T2*: The students who learned Scratch were more motivated and worked harder in solving more exercises compared with the other students in the class.

> *T2*: The students who were exposed to Scratch exhibited seriousness during the lessons compared with the other students. They met my demands and they knew all the working habits from the previous year. They were used to learning and practicing in teams. They were more active and they demonstrated this seriousness in the learning process from the beginning.

4.2.2. Self-Confidence. The students who had studied Scratch demonstrated higher self-confidence compared to the other students in the class. This was apparent from our observations of the satisfaction they exhibited after recognizing a newly introduced concept from their previous studies, as well as in their increased participation in the class. In particular, the observations showed that they were more inclined to answer questions posed by the teacher as compared to the other students:

S9: The students who were exposed to Scratch felt comfortable in the CS classes. We understood what the teacher taught and we felt an advantage over the other students in the class. The rest of the students were under pressure that they are behind.

S10: My classmates who did not learn Scratch were constantly stressed that we understand better than they do.

5. DISCUSSION

On the surface, the results may seem disappointing: many of the differences in the grades between the experimental group and the control group were not significant. However, even if we limit ourselves to these quantitative results, there are positive outcomes. For the concept of repeated execution (loops), the experimental group achieved significantly higher scores. Repeated execution has been known for a long time to be a difficult concept [Pea 1986]; if experience with Scratch improves learning "only" of difficult concepts, that would more than justify its use. We remark that our research checked just the intersection of concepts in Scratch and those studied in an introductory course using C# and Java. It would be interesting to see how experience with Scratch affects learning of truly advanced concepts, such as concurrency and event handling.

Similarly, the final test showed a significant result on the highest cognitive of Relational Creating. One expects students to achieve this level only after extensive experience, and we find it encouraging that studying Scratch in middle school can provide the necessary experience.

The qualitative results are encouraging as well. Teacher T2 reported a doubling of enrollment in secondary school CS. Therefore, even if there is no significant improvement in learning between the experimental and control groups, the students who learned Scratch were drawn from a larger population and we might have expected their average grades to be lower. This is consistent with the results of Ben-Bassat Levy et al. [2003], who showed that the use of the Jeliot program animation system primarily benefited the "middle third"—students who are capable of learning but not outstanding.

The teachers reported increased efficiency, finding that a concept that took six lessons to teach could now be taught in three. This has important implications for the teaching process: the teachers can explore more examples and devote more time to helping weaker students and to offering challenges to the stronger students. Even the restrictions that some Scratch students brought to learning introductory programming could be used as teachings moments to discuss programming language design at a basic level.

The higher levels of motivation and self-efficacy reported by the students should improve retention, because many students drop out for affective reasons [Kinnunen and Malmi 2006].

6. CONCLUSIONS

The results of this research provide strong evidence to justify learning CS in middle schools. The improvements in learning, teaching efficiency, and affective factors are easily discerned in the transition to secondary school CS.

Although we used Scratch because its simple two-dimensional world still supports advanced CS concepts, similar results should be observed when teaching with any environment that is adapted to middle school students. We believe that developers of such systems should carry out similar research projects to evaluate the contribution of each system to the transition to secondary- or tertiary-level CS.

The variable nature of the quantitative results points to a fertile field for future research. Why does Scratch improve the learning of some concepts and not others? What is the precise contribution of experience in Scratch to learning at the different cognitive levels? Detailed results of such research could make a significant contribution to CS pedagogical content knowledge at the middle school level and at higher levels as well.

APPENDIX: TEST QUESTIONS

The following questions were designed to assess the learning of the concept of bounded repeated execution on different cognitive levels:

***Question* 1:** Consider the following program segment [in C#]:

```
n1 = int.Parse(Console.ReadLine());
n2 = int.Parse(Console.ReadLine());
if (n1 < n2)
  for (int i = n2; i > = n1; i--)
  {
    Console.WriteLine(n2);
    n2++;
  }
else
 for (int i = n1; i > = n2; i--)
  {
    Console.WriteLine(n1);
    n1++;
  }
}
```

(a) [*Multistructural applying*] What is the output for the input n1 = 3, n2 = 5?
(b) [*Multistructural applying*] What is the output for the input n1 = 6, n2 = 3?
(c) [*Relational applying*] Give an example of input values for which the output is 2.
(d) [*Relational understanding*] Explain what the program segment does.
(e) [*Relational creating*] Write a program segment that gives the same results but that uses only one loop.

***Question* 2:** [*Relational creating*] Write a program segment that takes two integer numbers as input and prints the sum of all numbers between them (including the two numbers).

Example: For the numbers 4 and 8, the result will be 30 because $4 + 5 + 6 + 7 + 8 = 30$.

Example: For the numbers 3 and 6, the result will be 18 because $3 + 4 + 5 + 6 = 18$.

REFERENCES

Lorin W. Anderson, David R. Krathwohl, Peter W. Airasian, Kathleen A. Cruikshank, Richard E. Mayer, Paul R. Pintrich, James Raths, and Merlin C. Wittrock. 2001. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Addison Wesley Longman, New York, NY.

Michal Armoni, Tamar Benaya, David Ginat, and Ela Zur. 2010. Didactics of introduction to computer science in high school. In *Teaching Fundamental Concepts of Informatics*. Lecture Notes in Computer Science, Vol. 5941. Springer, 36–48. DOI:10.1007/978-3-642-11376-5_5

Tim Bell, Ian H. Witten, and Michael Fellows. 2005. Computer Science Unplugged. Retrieved November 24, 2014, from http://csunplugged.com/.

Ronit Ben-Bassat Levy, Mordechai Ben-Ari, and Pekka A. Uronen. 2003. The Jeliot 2000 program animation system. *Computers and Education* 40, 1, 1–15.

John B. Biggs and Kevin F. Collis. 1982. *Evaluating the Quality of Learning: The SOLO Taxonomy (Structure of the Observed Learning Outcome).* Academic Press, New York, NY.

Jerome S. Bruner. 1960. *The Process of Education.* Harvard University Press, Cambridge, MA.

Jerome S. Bruner. 1966. *Toward a Theory of Instruction.* Belknap Press, Cambridge, MA.

Lori Carter. 2006. Why students with an apparent aptitude for computer science don't choose to major in computer science. *SIGCSE Bulletin* 38, 1, 27–31.

Paul Curzon and Peter W. McOwan. 2008. Engaging with computer science through magic shows. *SIGCSE Bulletin* 40, 3, 179–183.

Wanda Dann, Stephen Cooper, and Randy Pausch. 2009. *Learning to Program with Alice* (2nd ed.). Pearson.

Wanda Dann, Dennis Cosgrove, Don Slater, Dave Culyba, and Stephen Cooper. 2012. Mediated transfer: Alice 3 to Java. In *Proceedings of the 43rd SIGCSE Symposium*. ACM, New York, NY, 141–146.

Elena Delgado-Rico, Hugo Carretero-Dios, and Willibald Ruch. 2012. Content validity evidences in test development: An applied perspective. *International Journal of Clinical and Health Psychology* 12, 3, 449–460.

Allison Druin and James Hendler (Eds.). 2000. *Robots for Kids: Exploring New Technologies for Learning*. Morgan Kaufmann.

Judith Gal-Ezer, Catriel Beeri, David Harel, and Amiram Yehudai. 1995. A high-school program in computer science. *Computer* 28, 10, 73–80.

Jens Gallenbacher. 2008. Abenteur Informatik. Retrieved November 24, 2014, from http://www.abenteuer-informatik.de/.

Barney Glaser and Anselm Strauss. 1975. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine, Chicago, IL.

Evelyn Jacob. 1998. Clarifying qualitative research: A focus on traditions. *Educational Researcher* 17, 1, 16–24.

R. Burke Johnson and Anthony J. Onwuegbuzie. 2004. Mixed methods research: A research paradigm whose time has come. *Educational Researcher* 33, 7, 14–26.

Caitlin Kelleher and Randy Pausch. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys* 37, 2, 83–137.

Päivi Kinnunen and Lauri Malmi. 2006. Why students drop out CS1 course? In *Proceedings of the 2nd International Workshop on Computing Education Research*. 97–108.

David J. Malan and Henry H. Leitner. 2007. Scratch for budding computer scientists. *SIGCSE Bulletin* 39, 1, 223–227.

Jane Margolis and Allan Fisher. 2003. *Unlocking the Clubhouse: Women in Computing*. MIT Press, Cambridge, MA.

Orni Meerbaum-Salant, Michal Armoni, and Mordechai Ben-Ari. 2011. Habits of programming in Scratch. In *Proceedings of the 16th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*. ACM, New York, NY, 168–172.

Orni Meerbaum-Salant, Michal Armoni, and Mordechai Ben-Ari. 2013. Learning computer science concepts with Scratch. *Computer Science Education* 23, 3, 239–264.

Orna Muller. 2005. Pattern oriented instruction and the enhancement of analogical reasoning. In *Proceedings of the 1st International Computing Education Research Workshop*. ACM, New York, NY, 57–67.

Seymour Papert. 2002. Hard Fun. Retrieved November 24, 2014, from http://www.papert.org/articles/HardFun.html.

Roy D. Pea. 1986. Language-independent conceptual "Bugs" in novice programming. *Journal of Educational Computing Research* 2, 1, 25–36.

Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: Programming for all. *Communications of the ACM* 52, 11, 60–67.

Rivka Taub, Michal Armoni, and Mordechai Ben-Ari. 2012. CS Unplugged and middle-school students' views, attitudes, and intentions regarding CS. *ACM Transactions on Computing Education* 12, 2, Article No. 8. DOI:10.1145/2160547.2160551.

Ursula Wolz, Henry H. Leitner, David J. Malan, and John Maloney. 2009. Starting with Scratch in CS1. *SIGCSE Bulletin* 41, 1, 2–3.

Received October 2013; revised May 2014; accepted May 2014