Learning Computer Science Concepts with Scratch

Orni Meerbaum-Salant Michal Armoni Mordechai (Moti) Ben-Ari
Department of Science Teaching
Weizmann Institute of Science
{orni.meerbaum-salant, michal.armoni,moti.ben-ari}@weizmann.ac.il

ABSTRACT
Scratch is a visual programming environment that is widely used by young people. We investigated if Scratch can be used to teach concepts of computer science. We developed new learning materials for middle-school students that were designed according to the constructionist philosophy of Scratch and evaluated them in two schools. The classes were normal classes, not extracurricular activities whose participants are self-selected. Questionnaires and a test were constructed based upon a novel combination of the Revised Bloom Taxonomy and the SOLO taxonomy. These quantitative instruments were augmented with a qualitative analysis of observations within the classes. The results showed that in general students could successfully learn important concepts of computer science, although there were some problems with initialization, variables and concurrency; these problems can be overcome by modifications to the teaching process.

Categories and Subject Descriptors
K.3.2 [**Computers & Education**]: Computer and Information Science Education - *Computer Science Education*.

General Terms
Human Factors, Experimentation.

Keywords
Scratch, Bloom's Taxonomy, SOLO taxonomy, middle schools, concurrency.

1. INTRODUCTION
In an attempt to increase interest in computer science (CS), much effort has gone into developing tools and activities for young people, primarily middle-school students, although these tools and activities have been used both by younger children and as preliminary learning materials in high schools and universities. Examples of such activities are Computer Science Unplugged [2] and magic shows [7]. Visual programming environments such as Scratch [24] and Alice [9] are very popular (see [14] for a survey of environments for novices). We chose to work with Scratch at the middle-school level because its simple two-dimensional world appeared sufficient to achieve our goals.

Visual programming environments facilitate the development of software in a context that is fun and non-threatening. The hope is that students will no longer feel anxiety and low self-esteem when faced with computers, so they will be more open to further study of CS. We are interested not only in such affective results, but also in investigating if the use of Scratch can facilitate the learning of CS concepts themselves. Our hope is that a positive answer will lead to better learning of CS when students see the same concepts again in high school or university.

We developed a set of Scratch-based learning materials aimed at middle-school students. Unlike previous studies that examined Scratch in extracurricular activities, our study was conducted in schools during normal school hours and were taught by middle-school teachers. The experimental setup was less than optimal in two aspects: While the course was not an elective one since every student in the two classes had to participate, it was not required by the curriculum, and thus the teachers and students probably felt less committed to this course, compared to required courses such as mathematics. In addition, we did not have sufficient time to fully train the teachers before the beginning of the school year. Nevertheless, we believe that the authentic in-school setting gives our research increased ecological validity.

The research was carried by using both quantitative and qualitative tools. Pre- and posttests and an interim test were designed and given to the students. They were based upon a novel combination of the Revised Bloom Taxonomy [1] and the SOLO taxonomy [4]. In addition, a researcher observed most of the classes and interviewed the students and the teachers.

In Section 2 we review previous work. Section 3 describes the learning materials. The research methodology, results and discussion follow in Sections 4 through 6, and Section 7 concludes the paper.

2. BACKGROUND
Interest in science and mathematics in general and CS in particular declines as students progress in their studies. The phenomenon has been studied, primarily from affective aspects such as attitudes [6]. The tools and activities mentioned above have been developed in order to encourage young people to maintain positive attitudes about CS as they pass into secondary schools and beyond. Lambert and Guiffre [17] examined the effect of CS Unplugged on students' *interest* in CS (rather than their views on CS). They found that the students were more interested in CS than before participating in CS Unplugged. Unfortunately, it has proved difficult to demonstrate that CS Unplugged actually achieves its wider long-term goals [27].

Visual programming environments have been used to attempt to achieve the same goals. Alice was found to improve college students' attitudes towards CS [22]. Storytelling Alice is a variant of Alice that enables users to create stories with interesting characters; middle-school girls using the system displayed greater

ICER 2010, August 9–10, 2010, Aarhus, Denmark.

motivation and willingness to spend extra time on the computer when compared with those using generic Alice [13].

Scratch (http://scratch.mit.edu), created by the Lifelong Kindergarten Group at the MIT Media Laboratory, is a media-rich system for novice programmers. Programs in Scratch are called *scripts* and they are created by dragging and dropping blocks that represent program components, such as Boolean expressions, conditions, loops and variables. Concurrent execution of all scripts begins when the interface's "green flag" is clicked. The environment eliminates syntax errors and gives immediate visual feedback through the behavior of the sprites. Scratch is augmented by a social computing network for sharing projects.

Maloney, Peppler, Kafai, Resnick and Rusk [21] reported on an experiment where Scratch was used by students in an after-school clubhouse. These students were self-selected and self-paced, receiving no formal instruction. By analyzing the students' projects, they found that the majority of the projects that actually constructed executable scripts used both sequential and concurrent execution. However, as the authors themselves note: "… *without realizing it*, most Scratch users make use of multiple threads" (p. 368, our emphasis). The researchers did not investigate if the use of concurrency demonstrates understanding of this concept. The internalization of concepts was measured by counting the portion of projects *using* them. These measures were higher (about 50%) for user interaction and loops, lower for conditional statements and for communications and synchronization (about 25%), and much lower for Boolean logic, variables and random numbers (about 10% or less).

Both Alice and Scratch have been used in universities in a CS0 course or at the beginning of a CS1 course. Scratch appears to help students understand programming by offering a different perspective than more traditional environments [31]. According to Malan and Leitner [20], students found Scratch exciting; they showed that it successfully familiarized inexperienced students with the fundamentals of programming. However, this research was based on surveys alone. In a panel session, Wolz, Leitner, Malan and Maloney [30] reported that after initially learning Scratch, the students' transition to Java or C appeared to be easier.

3. THE LEARNING MATERIALS

When a tool like Scratch is used in a formal school setting, it is important that high-quality learning materials be available so that teachers can follow a specific syllabus and not be required to develop every lesson from scratch (!). This is especially important for middle school teachers who tend to have a less advanced academic background in the subject matter.

By "learning materials" one normally means a textbook, but a routine textbook would not be compatible with the constructionist educational philosophy upon which Scratch is based [23]. This philosophy expands the constructivism learning theory by adding the need for an external concrete construction in learning processes to support mental construction. One of the main principles of Constructionism is learning-by-making. Our approach was as follows: We developed a textbook that explains everything we want the students to learn in full gory detail, but we do not expect students to read it; instead, it is intended as a guide for the teacher. The textbook provides a framework for constructing Scratch projects, for which we provide the full Scratch source code, although we leave it to the teacher to decide how much of this to give to the students.

The textbook is structured according to the following principles:

- Each chapter teaches a specific CS *concept* (rather than a Scratch feature) and the concepts are arranged in an order that we believe is best for learning about CS. In particular, we introduce conceptual material such as concurrency very early, postponing programming details like declaring variables as much as possible.
- Programming constructs are introduced *as needed*. For example, we do not introduce and explain all possible loop constructs in a single section, but rather introduce each one when it is needed. Of course the problems we pose are carefully selected so that their solutions "just happen" to need the construct we want to teach at that moment.
- The presentation is *project-based*. Each chapter is based upon a specific context such as animating dancers or a Pac-Man game. For each project, a set of tasks is presented to the learner: make the Pac-Man sprite open and close its mouth, make the sprite move, enable the user to control the direction in which the sprite moves. By constructing programs to implement the tasks one after another, the learner ends up with a working software artifact, while being taught concepts and programming constructs along the way.
- We *de-emphasize aspects of appearance* such as the costumes, sounds and visual effects. Instruction on these topics is included as optional tasks at the end of each chapter. This is an attempt to counteract the natural tendency of young people to engage in non-CS activities like drawing and playing music. For example, 21% of the students in the clubhouse used Scratch only for media manipulation [21].

We are not so naïve as to expect that our projects will be followed precisely as written. We are fully aware that students will want to build projects differently, and we think this is a positive phenomenon that encourages creativity. What we do hope is that the teacher will be able to guide the students' work in a way that will maintain the general framework of CS concepts.

The chapters of our learning materials are:

1. The Scratch environment; sequential computation; coordinates and directions; spatial initialization;
2. Concurrency in the form of multiple sprites;
3. Concurrency in the form of multiple scripts for a sprite; unbounded loops; simple conditions;
4. Communication and synchronization by sending and receiving messages; waiting;
5. Bounded and conditional loops;
6. Saving and recalling values in variables;
7. Conditional execution, events (other than receiving messages); randomness;
8. Numerical computation; accumulators and counters;
9. Lists;
10. Interference in concurrent programs;
11. Additional topics such as location sensors.

In the description of the experiment below, we note how much of this material was covered in each class.

4. RESEARCH METHODOLOGY

We studied novices who used the Scratch environment in middle schools. Our subjects were drawn from two classes: one consisted of 18 students (11 boys and 7 girls), while the other consisted of 28 students (all boys, studying in a boys-only school). The students in both classes were 14–15 years old (ninth grade). Each class took place in one two-hour period a week for one semester.

The teacher of the first class teaches mathematics in middle school and has no CS teaching experience, while the teacher of the second class has 15 years experience teaching CS. Both were encountering the Scratch environment for the first time. The teachers were given the teaching materials, as well as any technical and pedagogical support they needed, but the researchers did not intervene during the teaching of the classes.

We classify our research as *mixed method research* that combines qualitative and quantitative techniques, methods, approaches and concepts into one study [12]. A variety of data collection tools was used for triangulation to increase the validity of the findings.

4.1 Research Tools

Two groups of tools were used: The first group consisted of three tools – a pretest, a posttest, and an interim test. The pretest was given during the first lesson before the teachers started teaching. The interim test was given in the sixth lesson after the first five chapters had been studied (covering the concepts of initialization, bounded loops, conditional loops, message passing and concurrency). The posttest was given in the tenth lesson after two more chapters had been studied (Chapters 6 and 7), but the only additional concept in the test was that of variables. These three tests were carefully designed to allow a deep investigation of the internalization of CS concepts by students. We will elaborate on the design process of these three tools in Section 4.2.

The first author was a non-participant observer in both classes and collected the following materials, which constitute the second group of research tools:

- Field notes written during the observations.
- Interviews with the teachers. They were asked to reflect on special events, such as explaining difficulties in the students' understanding and the teaching process.
- The Scratch projects written by the students while working through the examples and the exercises of the learning materials.

In addition, interviews with students were also conducted; the analysis of the interviews will be reported elsewhere.

4.2 Combining Taxonomies for Research Design and Data Analysis

Various methods are traditionally used to assess students' learning in educational studies in general and in CS education research in particular. Two important methods are the Bloom taxonomy, in its original [5] or its revised form [1] (used for example in [29]) and the SOLO taxonomy [4] (used for example in [19]).

The revised Bloom taxonomy has two dimensions. The cognitive dimension includes six categories: Remembering, Understanding, Applying, Analyzing, Evaluating, and Creating, while the knowledge dimension includes four categories: Factual knowledge, Conceptual knowledge, Procedural knowledge and Metacognitive knowledge. The cognitive categories are ordered from simple to complex and from concrete to abstract, and each category is assumed to include lower ones; thus, these categories are traditionally considered to form an inclusive hierarchy. In the revised taxonomy the requirement of a strict hierarchy was somewhat relaxed to allow some overlapping between categories [15], but the general structure is still a hierarchal one.

The SOLO taxonomy has five ordered categories:

- Prestructural: Mentioning or using unconnected and unorganized bits of information which make no sense.
- Unistructural: A local perspective – mainly one item or aspect is used or emphasized. Others are missed, and no significant connections are made.
- Multistructural: A multi-point perspective – several relevant items or aspects are used or acknowledged, but significant connections are missed and a whole picture is not yet formed.
- Relational: A holistic perspective – meta-connections are grasped. The significance of parts with respect to the whole is demonstrated and appreciated.
- Extended abstract: Generalization and transfer – the context is seen as one instance of a general case.

We examined the SOLO taxonomy and the revised Bloom taxonomy for potential use in our study, both as tools to guide the design of the three tests and as data analysis tools. While the Bloom taxonomy was appealing as a potential design and analysis tool, we felt that in our context it would be too general and not informative enough. For example, Creating is considered to be much more complex than Understanding, but can we really say that creating a simple project – whose goal is to move one sprite from one point to another – is cognitively complex than fully understanding the concept of concurrency? We wanted to work with a strictly hierarchal taxonomy, enabling us to monitor students' progress, but one that matched the context of the study and its objectives. Intersecting the cognitive dimension with the knowledge dimension enriched the taxonomy, but we still felt that this enrichment did not cope with the difficulty just described.

In addition, the interpretation of the revised Bloom taxonomy for CS tasks is not straightforward [10, 28], and research shows that experts find it difficult to agree on an interpretation [11]. Lister and Leaney [18] claim that writing code can belong in different cognitive levels – Understanding, Applying, or Creating – depending on the magnitude of code, and, similarly, tracking code can belong to Remembering, Understanding or Analyzing, depending on the magnitude of the code. (We use the category names from [1]; Lister and Leaney used the original taxonomy.) Johnson and Fuller [11] argue that Applying is the main goal of CS practice and suggest adding a highest level of Higher Applying. Thompson et al. [28] interpret tracking code as Understanding, but they classify writing simple bits of code under Understanding as well. Fuller et al. [10] suggest a new taxonomy appropriate for CS education based on Bloom taxonomy, but in

which the cognitive dimension is restructured as a two-dimensional matrix differentiating between tracking code and writing code skills.

We felt that putting tasks of the same kind into different categories because of magnitude differences is not the proper solution, and that the issue of magnitude (and therefore of complexity) should be addressed explicitly.

The SOLO taxonomy seems to grasp another dimension – that of the holistic vs. the local perspective: being able to look at only one tree, seeing a few isolated trees, or seeing a whole forest. This categorization seems to answer the above problem in the sense that a full understanding of the concept of concurrency requires a holistic point of view, while creating a simple project requires only local knowledge. It also captures the magnitude issue, since tracking or producing small programs usually requires a relatively local perspective, while more complex code requires multistructural and relational abilities. The SOLO taxonomy is indeed very useful when analyzing a single activity such as analyzing the levels at which students track programs, or analyzing students' levels of program creation. However, using the SOLO taxonomy for various types of activities, simultaneously, is not straightforward.

Whalley et al. [29] used both taxonomies to analyze the same data, but this induced two independent analyses; however, the taxonomies are not totally independent. Thompson et al. [28] noted that a category of the Bloom taxonomy can sometimes reflect a few SOLO categories.

We chose to create a taxonomy which is a combination of the two. First, in the context of our study and corresponding to the learning goals, only a subset of categories seemed to be relevant. From the SOLO taxonomy we chose to focus on the three intermediate categories of Unistructural, Multisctructural and Relational. From the Bloom taxonomy we chose to focus on Understanding, Applying and Creating.

Our interpretation of these three Bloom categories in the context of CS education is as follows (using terms taken from the corresponding sub-categories of the cognitive dimension of the revised Bloom taxonomy [1]): *Understanding* is interpreted as the ability to summarize, explain, exemplify and classify CS concepts (among which are programming constructs), and to compare them. *Applying* is interpreted as the ability to execute algorithms or code: to track them and recognize their goals. *Creating* is interpreted as the ability to plan and produce programs or algorithms.

We combined these two sets of categories to form a taxonomy with three super-categories – Unistructural, Multistructural and Relational – each containing three sub-categories, corresponding to Understanding, Applying, and Creating. This yielded a nine-level taxonomy with the highest level of Relational Creating, and the lowest level of Unistructural Understanding.

The category Unistructural Creating, for example, describes the ability to create very small scripts doing one thing, or adding an instruction with local effect to an existing script. The category Relational Understanding describes the ability to fully understand a complex concept (such as concurrency), and coherently explain its various facets. The category Multistructural Applying describes the ability to track a program, capturing its various parts, but not the whole entity they form.

4.3 Using the Combined Taxonomy

The combined taxonomy was intended to be used both for designing the tools and for the analysis. In designing the tools, we aimed to construct questions that assess the various levels of the combined taxonomy. That is, questions for which a correct and full solution matches a specific category. Since the study population consisted of young students, we were limited in the length of the tests, and thus could not cover each of the categories in each of the three tests. Table 1 details the coverage of each of the categories in the three tests.

Table 1. Categories included in each test

	Unistructural			**Multistructural**			**Relational**		
	U	**A**	**C**	**U**	**A**	**C**	**U**	**A**	**C**
Pretest	X			X	X	X	X	X	
Interim test		X	X		X				
Posttest				X	X	X		X	X

The authors categorized the questions independently according to the combined taxonomy and then negotiated a consensus in cases of disagreement.

The three tests were designed to enable the analysis of the students' learning of the following CS concepts: initialization, loops (unlimited, bounded, conditional), message passing, variables and concurrency. Here are examples of questions corresponding to the some of the cognitive categories:

Unistructural

- Applying

  The following question is taken from the interim test. It refers to a script whose next to the last instruction is `point in direction …`.

  In which direction is the cat facing at the end of the script?

  a. -90 b. 0 c. 90 d. 180

Multistructural

- Understanding

  The following question is taken from the posttest.

  What is the difference between the instruction `say hello` *and the instruction* `broadcast hello`?

- Creating

  The following question is taken from the posttest. It continues the question given below for the Relational Creating level.

  Add to the scripts an instruction or instructions that will cause the scenario where the sprites change places to repeat itself five times.

Relational

- Applying

  The following question is taken from the posttest. It refers to a project with four sprites and seven scripts.

  Describe the behavior of the animals and the ball during the execution of all the scripts (after clicking on the green flag).

- Creating

  The following question is taken from the posttest.

  Construct an animation with two sprites. The sprites will be placed in two corners of the stage facing the center of the stage. Pick one sprite whose task will be to broadcast the message `switch` *to the other sprite. After the message is received the two sprites will change their places using the instruction* `glide 1 secs to x: 0 y: 0`. *During the process of changing places, the sprites will say something to each other when they meet.*

4.4 Data Analysis

The data were analyzed both quantitatively and qualitatively. Both types of analysis were based on the combined taxonomy described above.

In the quantitative analysis descriptive statistics were used to evaluate students' internalization of the various CS concepts, examining the number of students who gave a solution corresponding to a specific cognitive category. A t-test was used to examine whether there were any significant differences between the two classes.

The qualitative analysis used *content analysis*, which focuses on the actual content and the internal features of media. It is used to determine the presence of certain words, phrases, concepts, themes, characters or sentences within texts, and to quantify this presence in an objective manner. The text is coded into manageable analysis units and then examined using one of two basic methods: conceptual analysis or relational analysis [3]. The categories used were those induced by the combined taxonomy.

Data analysis was conducted in three steps corresponding to the three tests. In each step, the appropriate test was administered and the data collected from the tests and from observations conducted up until that time; these data were analyzed quantitatively and qualitatively. The intermediate findings were recorded and their results served also to refine the tool for the next step.

5. FINDINGS

The t-tests indicated that there was no significant difference between students' achievements in the two classes; therefore, for the descriptive statistics in this section we joined the two classes together into one group.

5.1 Internalization of CS Concepts

Both in the pretest and in the posttest we asked students to define CS concepts (in the posttest they could also add a corresponding Scratch instruction). Analyzing students' answers in the pretest, we saw that – as expected – the students did not have preexisting knowledge of CS concepts. In most cases, students did not give definitions for the given concepts, and when they did, the answers related to knowledge from another field, mainly from mathematics. For example, most students did not give a definition for the concept of a variable, but a few answers reflected its mathematical meaning: "a parameter", "a variable is x and x can be anything". Similarly, most students did not give a definition for concurrency, but those that did used an unrelated mathematical concept: "a rectangle with 4 parallel edges", "two parallel lines that can never meet each other".

Students' answers to this question in the posttest indicate that they internalized some CS concepts (Table 2). In Sections 5.3–5.6 we will elaborate on some of the concepts included in Table 2, using also the data from class observations and teacher interviews. Deeper analysis, using also student's answers to other questions, will be reported elsewhere.

Table 2. Students' internalization of CS concepts

Concept	Correct definition at the posttest
Initialization	17.5%
Bounded loops	57.5%
Conditional loops	75.0%
Message passing	62.5%
Variables	7.5%
Concurrency	7.5%

5.2 Achievements According to Cognitive Levels

The design of the three tests was guided by the combined taxonomy, with each question matching a cognitive category in the sense that a full solution to it satisfies this category. We grouped the questions in each test according to the different cognitive categories, and calculated the mean grades of students' solutions in each of the groups (Table 3, which is parallel in its structure to Table 1).

Students' grades were high for questions corresponding to lower cognitive categories (the three left columns). The mean grade of the Relational Creating category in the posttest was quite disappointing (32.8), while the mean grade for the Multistructural Creating level was higher, but still much lower than that of the Relational Applying level.

Figure 1 shows the change in the students' mean grades from the pretest to the posttest in the four categories that appeared on both tests: Multistructural Understanding, Applying and Creating, and Relational Applying (N=40). We expected to see much higher grades in the posttest, and indeed this is the case for 3 categories. For the Multistructural Applying category (for which we also have an interim grade) there was no significant change (t= −0.49).

Table 3. Students' Cognitive Performance (SD in italic)

	Unistructural			Multistructural			Relational		
	U	**A**	**C**	**U**	**A**	**C**	**U**	**A**	**C**
Pretest N = 46	84.8 *31.4*			6.5 *11.4*	55.6 *26.8*	0 *0*	3.3 *12.5*	22.3 *28.3*	
Int. test N = 43		77.9 *31.4*	81.4 *29.4*		55.2 *29.4*				
Posttest N = 40				54.8 *25.9*	45.6 *43.8*	50.6 *39.8*		72.5 *39.1*	32.8 *25.8*

This surprising finding can be explained by the fact that the corresponding sets of questions were very different in nature. The questions corresponding to the Multistructural Applying category in the pretest required the students to track algorithms that were presented verbally and that involved simple calculations, whereas in the interim test and the posttest, the corresponding questions involved tracking Scratch programs that explicitly referred to spatial orientation (coordinates), a skill that may be difficult for students of this age.

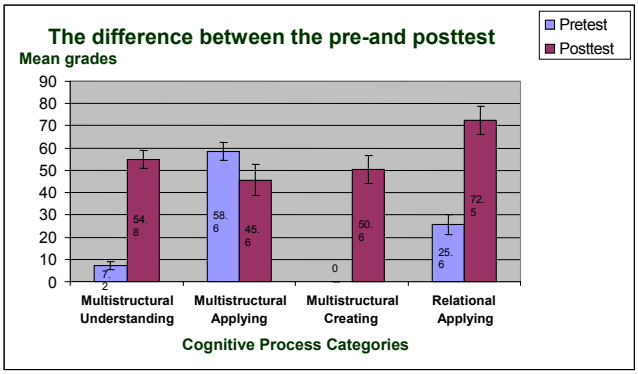

Figure 1. Pre- and posttests in terms of cognitive performance

5.3 The Concept of Initialization

Initialization is known to be a difficult subject since it is hard for students to make assumptions about the initial state of a system, a skill that requires more general cognitive activities beyond programming [26]. In the context of Scratch, the concept of initialization is not solely connected with variables; one usually has to initialize the position and direction of sprites. Many students could not give a satisfactory definition for this concept in the posttest, and seemed to confuse the concept of initialization with that of starting the execution by clicking the green flag in the Scratch interface. Although this could indicate that they did not understand the concept of initialization, it is possible that this resulted from a simple linguistic confusion. Since the teachers usually presented concepts without explicitly naming and defining them, the students may have confused the words "start" and "initialization", which are very similar in Hebrew. The qualitative analysis of other questions in the posttest in which initialization was used, as well as of the class observations, showed that students' internalization of initialization was better than what was reflected by the definitions they gave.

5.4 The Concept of Loops

The concept of loops is a basic one, but it is reported as difficult for students [8]. Our findings indicate that the students internalized this concept: In the posttest 57.5% of the students correctly defined bounded loops and 75% correctly defined conditional loops. Our learning materials introduce the concept of loops relatively early in the form of infinite loops, which are very common in Scratch projects. We believe that this early experience with one loop construct facilitates the successful learning of other types of loops.

5.5 The Concept of Variables

The variable concept is also considered as a difficult one for students [16]. In our learning materials this concept is introduced in Chapter 6. Unlike many other CS concepts which appear in an unusual guise in Scratch, the support for variables in Scratch is quite similar to that in regular programming languages: One has to declare a variable (as either global or local to a sprite) and then assign values using blocks such as `set` or `change`. Values are read using reporter blocks that hold variable name. It seems as if this concept is less likely to be acquired by self-learning in a trial-and-error manner. We believe that *explicit* instruction of the concept is needed, but for reasons relating to the teachers' background and time constraints, such instruction was not available to the extent required. The mathematics teacher introduced this concept to her students by saying: "a variable, as you already know from mathematics, can be assigned a value", while the CS teacher merely said "it is something that contains data which we can use". Internalization of concepts often requires adequate exposure to several instances [8], so we believe that with adequate instruction, the students' internalization of this concept can improve.

5.6 The Concept of Concurrency

One of the most attractive features of Scratch is the natural way that it encourages the use of concurrency from the very beginning. We were interested in investigating the extent of the students' internalization of the concurrency concept.

Concurrency appears in the learning materials already in Chapter 2. We divided this concept in two: *Type I concurrency* occurs when several sprites are executing scripts simultaneously, such as sprites representing two dancers. *Type II concurrency* occurs when a single sprite executes more than one script simultaneously; for example, a Pac-Man sprite moving through a maze while opening and closing its mouth.

According to our findings, Type I concurrency seems to be much more intuitive for students and easier to grasp. Although no one could give a verbal definition for concurrency in the pretest, the pretest included one question in which a Type I concurrent scenario of a relay race was described. About 26% of the answers demonstrated some comprehension of the type of concurrency. This value is not high, but we should remember that it was measured before the teaching process began.

Type II concurrency was introduced in Chapter 3 soon after Type I, and students had problems with it when they solved the textbook assignments. We observed students programming the instructions in a sequential order instead of concurrently for a single sprite, and students neglecting to include the instruction <when green flag clicked> that causes all the scripts to be executed, including it only at the beginning of one of the scripts.

Four questions dealt with concurrency concepts in the posttest, each testing a different cognitive category (Table 4). We can see that students' achievements in the two questions involving Type II concurrency are low, while their achievements in the other two questions are higher. Comparing the results for the second and third question led to a puzzling finding. We conjectured that students' performance should decrease as the level of the cognitive category increases. However, this was not the case here: 62.5% of the students were able to answer correctly the third question, categorized at the Relational Applying level, while only 37.5% answered correctly the second question, categorized at the Multistructural Applying level. We will discuss this further in Section 6.

Table 4. Internalization of the concurrency concept in the posttest

Question	Cognitive category	Concurrency Type	Correct answers
1	Multistructural Understanding	I,II	7.5%
2	Multistructural Applying	I	37.5%
3	Relational Applying	I	62.5%
4	Relational Creating	I, II	2.5%

In the posttest, when asked to give a definition of concurrency (the first question) only 7.5% of the students gave a definition that referred to both types of concurrency. 22.5% of the definitions were limited to Type I concurrency and 42.5% were limited to Type II concurrency and 27.5% did not give any definition. Even those students whose definition reflected Type II concurrency did not succeed in writing a program using Type II concurrency (the fourth question); only one student answered this question correctly.

6. DISCUSSION

Our findings showed that a meaningful learning process occurred as a result of the course. The students demonstrated internalization of certain CS concepts and an improvement in their achievements in terms of cognitive performance.

However, we found that the students had problems with three concepts: initialization, variables and concurrency. These concepts are more abstract than other concepts that we investigated. One possible explanation is that these three concepts involve several Scratch instructions and their relationships, whereas the other concepts can be expressed in Scratch by one instruction. Therefore, in the terms of the SOLO taxonomy, their internalization – even to a minimal extent – necessitates at least multistructural abilities, and, in some cases, even requires relational abilities, while internalization of other concepts can be achieved by local, unistructural abilities.

We believe that appropriate instruction can improve the internalization of the three concepts. In our case, the teaching process was less than optimal. Despite its flashy exterior, Scratch is a sophisticated software system and it takes time to learn its technical and pedagogical aspects. The teachers themselves reflected on this issue. For example, the CS teacher said that normally she prepares a course during the summer vacation, but in this case she was asked to teach this course after the school year had begun. We are now designing a course for in-service teachers on the didactics of teaching CS concepts using Scratch to be given before the next school year. We believe that this will result in an improved teaching process.

The design of the research tools and the data analysis used a taxonomy that combined the Bloom and SOLO taxonomies. Our findings indicate that this combined taxonomy captures cognitive characteristics of CS practice. Yet, at lease one anomaly was observed: Achievements in the high cognitive category of Relational Applying were much higher than in the lower category of Multistructural Creating. Tracking code (Applying) is traditionally considered easier for students than creating code [10, 19, 25], but according to the rationale of our taxonomy, the complexity of the code, expressed in the SOLO abilities it requires (Multistructural or Relational), is more significant. A possible explanation is that the categories are not really discrete with strict boundaries, but rather allow a continuum of development between cognitive categories. For questions in a "boundary area", other factors such as the type of task or the concepts it involves, may overrule the effect of SOLO complexity.

7. CONCLUSION

We investigated the use of Scratch to teach concepts of computer science. We developed new learning materials based upon the constructionist philosophy of Scratch, and evaluated their use in middle-school classrooms. The results showed that most students are able to understand CS concepts, thus supporting the claims of Scratch to be a viable platform for teaching CS. Specific difficulties that we encountered in teaching initialization, variables and concurrency can be ameliorated by improving the explicit teaching of these concepts.

Our research used a novel combination of the Revised Bloom's taxonomy and the SOLO taxonomy. We can recommend such an integration of taxonomies in order to achieve a research framework that is applicable to the specific needs of CS education research. This combined taxonomy should be further investigated in various, perhaps wider contexts, and possible refinements (such as relaxing its discrete nature) should be examined.

8. ACKNOWLEDGMENTS

This research was partially supported by the Israel Science Foundation (grant 1277/09). We would like to thank Fatima Hallack for her help with this research.

9. REFERENCES

[1] Anderson, L. W., Krathwohl, D. R., Airasian, P. W., Cruikshank, K. A., Mayer, R. E., Pintrich, P. R., Raths, R. and Wittrock, M. C. Eds. 2001. A Taxonomy for Learning, Teaching and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives. Addison-Wesley Longman.

[2] Bell, T., Witten, I.H., Fellows, M. 2005. Computer Science Unplugged. http://csunplugged.com/ (last accessed 20 April 2010).

[3] Berelson, B. 1952. Content Analysis in Communication Research. Free Press.

[4] Biggs, J. B. and Collis, K. F. 1982. Evaluating the Quality of Learning: The SOLO Taxonomy (Structure of the Observed Learning Outcome). Academic Press.

[5] Bloom, B. S., Mesia, B. B., and Krathwohl, D. R. 1964. Taxonomy of Educational Objectives (2 volumes: The affective domain; The cognitive domain). David McKay.

[6] Carter, L. 2006. Why students with an apparent aptitude for computer science don't choose to major in computer science. SIGCSE Bull. 38, 1, 27-31

[7] Curzon, P. and McOwan, P. W. 2008. Engaging with computer science through magic shows. SIGCSE Bull. 40, 3, 179-183.

[8] Dancik, G. and Kumar, A.N. 2003. A tutor for counter-controlled loop concepts and its evaluation. In Proceedings

of the Frontiers in Education Conference (Boulder, CO, November 05-08, 2003). Session T3C (no page numbers).

[9] Dann, W., Cooper, S. and Pausch, R. 2009. Learning to Program with Alice, Second Edition. Pearson.

[10] Fuller, U., Johnson, C. G., Ahoniemi, T., Cukierman, D., Hernán-Losada, I., Jackova, J., Lahtinen, E., Lewis, T. L., Thompson, D. M., Riedesel, C., and Thompson, E. 2007. Developing a computer science-specific learning taxonomy. SIGCSE Bull. 39, 4, 152-170.

[11] Johnson, C. J. and Fuller, U. 2006. Is Bloom's taxonomy appropriate for computer Science? In Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006 (Uppsala, Sweden, November 09-12, 2006). ACM, New York, NY, 120-123.

[12] Johnson, R. B. and Onwuegbuzie, A. J. 2004. Mixed methods research: A research paradigm whose time has come. Educational Researcher, 33, 7, 14-26.

[13] Kelleher, C., Pausch, R., and Kiesler, S. 2007. Storytelling Alice motivates middle school girls to learn computer programming. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (San Jose, California, USA, April 28 - May 03, 2007). ACM, New York, NY, 1455-1464.

[14] Kelleher, C. and Pausch, R. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv.* 37, 2, 83-137.

[15] Krathwohl, D, R. 2002. A revision of Bloom's taxonomy: An overview. Theory into Practice 41, 4, 212-264. http://www.unco.edu/cetl/sir/stating_outcome/documents/Krathwohl.pdf (last accessed 20 April 2010).

[16] Kuittinen, M, and Sajaniemi, J. 2004. Teaching roles of variables in elementary programming courses. SIGCSE Bull. 36, 3, 57-61.

[17] Lambert, L. and Guiffre, H. 2009. Computer science outreach in an elementary school. J. Comput. Small Coll., 24, 3, 118-124.

[18] Lister, R. and Leaney, J. 2003. Introductory programming, criterion-referencing and Bloom. SIGCSE Bull. 35, 1, 143-147.

[19] Lister, R., Simon, B., Thompson, E., Whalley, J.L. and Prasad, C. 2006. Not seeing the forest for the trees: Novice programmers and the SOLO taxonomy. SIGCSE Bull. 38, 3, 118-122.

[20] Malan, D. J. and Leitner, H. H. 2007. Scratch for budding computer scientists. SIGCSE Bull. 39, 1, 223-227.

[21] Maloney, J., Peppler, K., Kafai, Y., Resnick, M., and Rusk, N. 2008. Programming by choice: Urban youth learning programming with Scratch. SIGCSE Bull. 40, 1, 367-371.

[22] Moskal, B., Lurie, D. and Cooper, S. 2004. Evaluating the effectiveness of a new instructional approach. In Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education (Norfolk, VA, March 03-07, 2004), ACM Press, New York, NY, 75-79.

[23] Papert, S. and Harel, I. 1991. Constructionism. Ablex.

[24] Resnick, M., Maloney, J. Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y. 2009. Scratch: Programming for all. Commun. ACM, 52, 11, 60-67.

[25] Robins, A., Rountree, J., and Rountree, N. 2003. Learning and Teaching Programming: A Review and Discussion. Computer Science Education, 13, 2, 137-172.

[26] Samurçay, R. 1989. The concept of variable in programming: its meaning and use in problem-solving by novice programmers. In E. Soloway and J. C. Spohrer (Eds.). Studying the Novice Programmer, Lawrence Erlbaum Associates, 161-178.

[27] Taub, R., Ben-Ari, M., and Armoni, M. 2009. The effect of CS Unplugged on middle-school students' views of CS. SIGCSE Bull. 41, 3, 99-103.

[28] Thompson, E., Luxton-Reilly, A., Whalley, J. L., Hu, M., and Robbins, P. 2008. Bloom's taxonomy for CS assessment. Proceedings of the Tenth Conference Australasian Computing Education (Wollongong, NSW, Australia, January 20-23, 2008), 155-161.

[29] Whalley, J., Lister, R., Thompson, E., Clear, T., Robbins, P., and Prasad, C. 2006. An Australasian study of reading and comprehension skills in novice programmers, using the Bloom and SOLO taxonomies. In Proceedings of the Eighth Australasian Conference Computing Education (Hobart, Australia, January 16-19, 2006), 243-252.

[30] Wolz, U., Leitner, H. H., Malan, D. J., and Maloney, J. 2009. Starting with Scratch in CS1. SIGCSE Bull. 41, 1, 2-3.

[31] Wolz, U., Maloney, J., and Pulimood, S. M. 2008. 'Scratch' your way to introductory CS. SIGCSE Bull. 40, 1, 298-299.