Scratch vs. Karel – Impact on Learning Outcomes and Motivation

Alexander Ruf
TUM School of Education
Technische Universität München
Arcisstr. 21
80333 München, Germany
rufa@in.tum.de

Andreas Mühling
TUM School of Education
Technische Universität München
Arcisstr. 21
80333 München, Germany
andreas.muehling@tum.de

Peter Hubwieser
TUM School of Education
Technische Universität München
Arcisstr. 21
80333 München, Germany
peter.hubwieser@tum.de

ABSTRACT
This paper presents the results of an experiment regarding the effects of using one of two different programming environments in secondary schools. Both “Scratch” and “Karel the Robot” have been successfully used in these settings previously. These two environments are also representative for two classes of programming environments for beginners. One is more graphically oriented and may therefore alleviate the steep learning curve of programming while the other is text-based and therefore more akin to “real” programming. Also, one places more emphasis on the visualization of program structure and the other emphasizes visualizing program flow.

The experiment has been conducted in parallel in two school classes, each using one of the two approaches. The abilities of the students were tested before and after the experiment as well as their intrinsic motivation and the perceived self-regulation. The results show, that the class using Scratch has higher intrinsic motivation and performs better, however the Karel class shows a higher identified regulation.

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

General Terms
Algorithms, Experimentation, Human factors, Languages, Measurement

Keywords
Algorithms, bebras, computer science education, concept maps, concepts, control flow statement, karel, karol, motivation, novice programmer, programming, programming environment, self-regulation, scratch

1. INTRODUCTION
The times when learning how to program was a venture mostly undertaken by self-motivated kids who were happy when seeing a “Hello world” appearing on a green screen are long gone. In today’s computer dependent world, CS education is more and more part of general – especially secondary – education and many students, whether intrinsically motivated for it or not, have to attend programming courses. Consequently, didactic tools have been developed that support the rather steep learning curve. Often, these tools are helping to make the “syntax barrier” less burdensome for students and providing some kinds of graphical capabilities that offer “more” then mere text-console output.

In general, the focus of visualization of such tools can lie on two aspects:

1. Visualizing the program flow.
2. Visualizing the program structure.

Also, there are two possible types of how the program is created in the first place:

1. Entering program text directly (often using a simple syntax).
2. “Clicking” the program together in a GUI from predefined building blocks.

While in theory these two aspects of how the program is created and what is visualized are independent, when looking at several classic representatives of such tools, there are only two contrasting approaches that can be identified.

This first approach is based on the first option of both the visualization and the program creation: A student enters program code into some form of editor – similar to typing code in a “real” programming language. Then, this code is run step-by-step. This – together with a suitable form of “output” – allows the student to easily observe the program flow. Entering code may be guided by the mouse, code-completion, or features like indentation or syntax highlighting. The syntax is typically designed in a way that makes it easy to learn and understand. One of the classics of these tools is Karel the Robot. Even though it has been improved and adapted to current systems (e.g. Java Karel), the basic idea and conception remains the same: A student writes code that operates a moving robot in a grid-based world (see next section for more details). The Turtle graphics of Logo are very similar.
In contrast, a different and somewhat more recent didactic approach is based on the second options of the two choices given above: “Graphical” programming languages like Scratch or Alice. They differ, usually, in two respects: First, the visualization of the actual program flow is of less importance and second, there is no syntax that has to be mastered. The program elements are only “clicked” together to build a program – which in turn places more emphasis on visualizing the (syntactic) program structure. With syntax errors having been rendered impossible, the learning curve has been made a lot less steep. It is possible even for primary school students to create programs in this way. However, the “programming” is a lot different from programming in a real programming language.

The research question that underlies the experiment presented in this paper is: How does the visualization of program flow combined with “actual” written program code (albeit in a simplified form) affect the abilities and motivation of beginning programmers compared to an approach that visualizes the program structure, is syntax-less and more oriented towards creating visible effects with programs.

To this end, two classes in an otherwise identical educational setting were instructed using Scratch and Karel as representative tools for the two approaches. The change in abilities, motivation and self-regulation were measured.

2. BACKGROUND AND RELATED WORK

This section presents related work of several aspects relevant for our experiment. This entails the educational setting in which it took place, the two programming environments that were used and prior research results regarding these environments.

2.1 Computer Science Education in Bavaria

In the German federal state of Bavaria, “Informatics” is a compulsory subject in the “Gymnasium” which is the highest type of secondary school. Details about the conception and introduction of the subject have been published internationally, e.g. [11] or [23]. The subject starts in grade 6, which is the second year of Gymnasium. This paper is mostly concerned with grade 7. Each year, there are more than 40,000 students attending Informatics education in grade 7

1. The curriculum for the 7th grade mostly deals with algorithms and their fundamental “building blocks” statement, sequence, conditional statement and loop. A curriculum in Germany is mandatory for the teachers, i.e. they are not free to decide if or to what extent they are following the curriculum in their classes.

2.2 Robot Karol

According to the curriculum, the algorithms must be implemented in a programming environment suitable for the age (typically 12 years old) of the students that visualizes the single steps of the execution. While there are several different environments that could be used, usually, in Bavaria, a German version of “Karel the Robot” [26], called “Robot Karol” 2 is employed by the teachers. Since other tools are

Figure 1: Screenshot of Robot Karol.

Robot Karol is a simulated robot, who “lives” in a grid-based world of fixed dimensions (see Fig. 1). Leaving out some advanced and specific operations, the robot can be operated as follows: It can move in every direction, turn by 90 degrees left or right, place and pick-up bricks in front of it, and create or remove markers on its current grid cell. Also, the typical algorithmic control structures, “sensor” input (e.g. current grid is empty) and some logical functions can be used in order to create programs which let Karol solve certain tasks – like finding the way out of a labyrinth or following a path of bricks. Programming is done by entering text in an editor. This process is supported by a context menu offering code fragments – similar to code completion in modern IDEs – and is very similar to “real” programming.

2.3 Scratch

In contrast to this approach, there are tools like Scratch, which use graphical symbols that represent the several control structures. The “programming” is done by dragging-and-dropping these together like a puzzle (see Fig. 2). Scratch has been published in 2004 [18] and is used frequently throughout the world [27]. For more details about Scratch as a programming language and environment see [19].

2.4 Former Studies

Scratch is successfully used all over the world as a programming environment for beginning programmers. Especially for “middle-years” students (typically age 8 to 15), who are also the focus of this paper, many curricula and CS-courses are based on using Scratch. For example, see [9] and their work for “exploring computer science”3 or the MyCS curriculum4 presented in e.g. [32]. In [10] Scratch was used on a similar age group as in this paper (age 12 to 13) to assess the computational learning of the students, who have learned the basics of programming in a 6 week module using Scratch.

However, in contrast to the experiment in this paper, most prior experimental studies do not investigate the learn-
ing success of using Scratch in a regular school class. Instead, typically, either voluntary courses e.g. at a “computer clubhouse” [20] or at an “interdisciplinary two-week summer camp” [8] or courses for young adults, like an introductory course for future teachers [7] are investigated. Also, there is a wide variety of the actual goal that Scratch is employed for. In [29] for example, it is used in order to foster creativity during programming. The study presented in [5] is set in a “classroom writing-workshop” and investigates how “fundamental CS concepts as well as the wider connection between programming and writing as interrelated processes of composition” are learned by the participants. Scratch has also been used for developing a model for the “Progression of Early Computational Thinking (PECT)” [33]. All studies show that Scratch is a useful tool for programming beginners and that its application has a positive effect on learning.

A study that employs a similar setting to our experiment, i.e. regular school classes, is presented in [21]. It is investigated whether or not “Scratch can be used to teach concepts of computer science”. The result is, again, positive, albeit the students had problems with the three concepts of “initialization, variables and concurrency” – which are excluded in our experiment. Similar to our experiment in other aspects is the study presented in [16]: It compares Scratch to a more text-based programming environment, namely “Logo”. Except for the concept of conditional statement, there were no observable advantages of using Scratch. In contrast, the Logo users “had on average higher confidence in their ability to program”.

Karel the Robot has originally been deemed an ideal tool for programming beginners. The use of robots – even if only simulated ones – was assumed to be motivating as shown in e.g. [15]. Also, the use of “mini-languages” has been seen as a viable way to learn programming [3]. Karel the Robot is a typical representative of such a mini-language. The “Taxonomy of Programming Environments and Languages for Novice Programmers” [13] lists Karel under the category “making programming concrete”. Nowadays, most publications are based on the Java versions of Karel the Robot, e.g. [4] or [2]. These have been used, among others, in CS1 courses successfully [1]. An overview over the most widely used Java versions of Karel can be found in [13]. In [14], Karel is used to investigate problem solving strategies of algorithmic problems – in the very same educational setting (7th grade of Bavarian secondary schools) as the experiment of this paper. Finally, in [35], Karel has been used in an introductory programming course in parallel with Python. Karel was used as a visualization of the programming concepts – however this concept was not successful, probably due to the lack of common elements between Karel and Python.

3. METHODOLOGY

This section describes both the educational setting that the experiment was conducted in and the data that was collected from it.

3.1 Description of the Setting

This paper presents the results of an experiment conducted in two classes of the 7th grade of a Bavarian Gymnasium. Both classes were held parallel and taught by the same teacher for half of the school year, from September 2013 to February 2014. In this time, the relevant part of the curriculum (algorithms, as described above) was taught. The lessons were planned exactly alike for both classes, the only difference being that one class used Robot Karol and the other used Scratch as the programming environment of choice to introduce the basics of programming. We will call them “Scratch class” and “Karol class” from now on. As the students have not been randomly assigned to one of the two possible “treatments” it is not a real “experiment” in the statistical sense. However, as presented and discussed in the next sections, we did look for differences in the relevant prior knowledge between the two groups and did not find any.

Informatics education is comprised of one lesson (45 minutes) per week. Both classes had their lesson on the same weekday one class directly following the other. Both classes were of roughly equal size: The Scratch class consisted of 29 students (16 girls and 13 boys), the Karol class consisted of 27 students (17 girls and 10 boys). All students have been between 12 and 13 years old. The teacher has been teaching Informatics since the subject was introduced in Bavaria. In particular, he has often taught the 7th grade in prior years both using Karol and using Scratch.

In total, the experiment ranged over 18 lessons, including lessons that were used for conducting tests (see below). The lessons are based on a constructivist design, meaning that the students are most of the time actively engaged in problem solving. They are typically trying to solve tasks that the teacher presents to them. Necessary prior knowledge is presented briefly at the beginning. Apart from that, the role of the teacher is mostly supporting and guiding the students in their problem solving attempts. Lesson planning therefore is “reduced” to finding or creating suitable exercises. Suitable in this case means that the exercises should be solvable equally well using either Karol or Scratch. Therefore – and since the lessons should be as alike as possible – not all features of Scratch were used. Mostly, especially in the beginning, only one animated figure was used in the exercises. Also, the exercises must in general incorporate some newly introduced concepts in a way that solving them without the new concepts is either impossible or way more difficult.

As part of their learning material, the students received a four page handout that they could use as a reference for the most important concepts. Even though not explicitly stated in the curriculum, both classes were taught a graphical representation of control structures in the form of Nassi-Schneiderman diagrams. After each new control structure
that the students learned, the corresponding diagram was presented to them and trained with exercises that asked them to translate between the two representations “code” and “diagram”. The Nassi-Schneiderman diagrams are introduced in this early stage as a basis for the subsequent classes. They are then used to help introducing other programming languages like Java. We use the diagrams for comparison between the two groups in the analysis presented below, as it is a representation of program code that both groups have learned.

The following list shows how the lessons progressed chronologically and in which order the topics were presented to the children:

1. Getting to know the programming environment (Scratch or Karol).
2. Implementing programs, sequence of operations.
3. Loops with fixed number of iterations.
5. Loops with conditions.
7. A bigger programming task.

The concept of variables has been left out on purpose, as it is not part of the curriculum of the 7th grade. It is introduced in higher grades. Also, specific elements of the programming environment like playing sounds or more design oriented elements have been left out of the lessons and the handouts. Nevertheless, students were obviously encouraged to try these elements on their own, if they desired.

3.2 Data Collection

The data that was collected in the course of the experiment consists of a pre- and post-test of the students’ abilities, a graded exam, a survey, and concept maps.

3.2.1 Pre- and Post-Test

The goal of the tests was to measure a difference in programming abilities before and after the 18 lessons. Since “abilities” is a rather general term and the students are at the very beginning of their programming education, we needed a test that consists of exercises that deal with algorithmic control structures. Instead of designing such a test on our own, we were looking for established exercises or tests that can readily be employed and have been used before by many others. We opted for exercises from the international Bebras contest. They seem perfect, as they are all fulfilling the following criteria:

- representing informatics concepts,
- easily understandable,
- solved within 3 minutes,
- short, e.g. presentable at a single screen page,
- solvable at computer without use of other software or paper and pencil,
- independent from specific systems,
- interesting and/or funny.

In 2013, more than 700,000 students from 29 nations took part in the Bebras contest. Depending on their age, the students have to solve different task sets which consist of tasks of three different levels of difficulty (easy, medium, and hard). The tasks are covering more topics than programming, therefore we selected several that are specifically dealing with algorithmic thinking, or which are easier to solve if control structures are known. Also, we made sure that the tasks are applicable with regard to their prior education in programming. For example, tasks that in which variables or the concepts of state appears have not been used due to the contents of the curriculum, as described above. The selected tasks have all been taken from the German versions of the Bebras contests from 2010 to 2012. Two examples from 2011 respectively 2010 that was used in the test can be seen in Fig. 3. and Fig. 4.

Schildkröten

Du hast einen Schildkrötenroboter zum Geburtstag bekommen, der folgende einfache Anweisungen ausführen kann:

- Drehe dich um 90 Grad nach rechts
- Drehe dich um 90 Grad nach links
- Fahre 30 Zentimeter vorwärts

Der Schildkrötenroboter ist so gemacht, dass er eine ihm gegebene Folge von Anweisungen solange wiederholt, bis man ihn ausschaltet.

Welche Folge von Anweisungen lässt den Schildkrötenroboter ein Quadrat fahren?

A)  
B)  
C)  
D)  

Figure 3: One example task of the Bebras contest in German: A turtle has three possible operations (turn 90 degrees to the right or left, go 30 cm forwards). The turtle will repeat its sequence of operations indefinitely. The task asks for the sequence of operations that will make the turtle run in a square.

In total, nine tasks have been selected for the tests, three of each level of difficulty. Each task is a multiple choice question with four options and exactly one right choice. The students of the two classes were presented with the same test once before the first and once after the last lesson of the experiment. The students of both classes did not participate in the Bebras contest themselves, so the tasks were completely new to them. Pre- and post-test were identical. Since the students did neither receive any solution or explanation of the solution, nor their own result and there were about 18 weeks between the tests, an improvement in the results it very likely not influenced by the fact that the students already knew the questions. There was no time limit given in the tests. The students were told to only give an answer if they are certain, since a wrong answer incurs a score penalty, however the score achieved by the students didn’t have any effects on their grade.
Figure 4: Another example task of the Bebras contest in German: A beaver has three possible “activities”.

- “running” calls “running around the block” three times,
- “running around the block” in turn calls “running along the street” four times, and
- “running along the street” calls two basic operations “take 100 steps” and “turn 90 degrees to the left”.

The question is how many steps the beaver has made after having performed the activity “running” once.

### Graded Exam

To fulfill the requirements of the curriculum, a (short) graded exam had to be taken by the students. The results were analyzed as part of the experiment. The exam was held at the end of the 18 weeks sequence (before the post-test). The date were told the students one week in advance, so they could prepare for the exam. It consisted of two problems: One had to be solved using pen and paper and the other had to be solved using a computer.

1. The first task was to describe in own words the effect of a given program (using all the control structures, also nested) and to give a representation of the program in a Nassi-Schneiderman diagram.
2. The second task presented a problem in text form and asked the students to solve it by writing a program. The class that used Scratch received the following problem:

   “A spider wants to find its way out of a labyrinth. It has marked the intersections beforehand, so that it knows how to turn. If an intersection is marked in red, the spider has to turn left. If it is marked yellow, it has achieved its goal (see Figure below)”.

The exam was kept very similar between both classes and, as judged by the teacher, of identical difficulty. Also, it was scored identically for both classes and graded using an identical scheme.

### Surveys

Apart from the abilities of the students, the effect that the lessons have on their intrinsic motivation and their perceived self-regulation were of interest. To investigate this, the students of both classes were given a survey at the end of the experiment that consisted of two major parts.

The part that deals with the intrinsic motivation was taken from [34]. There, a short scale of intrinsic motivation is presented and tested, based on the “Intrinsic Motivation Inventory” (IMI) of Deci and Ryan, “a multidimensional measurement device intended to assess participants’ subjective experience related to a target activity in laboratory experiments”.

The short scale is comprised of four subscales dealing with
- interest/enjoyment,
- perceived competence,
- perceived choice, and
- pressure/tension.

Instead of the 22 items of the IMI, the short scale only uses 12 items. Also, the short scale has already been translated into German and tested using students of a similar age. It has been found to be objective, reliable and valid. A 5-point Likert scale is used.

In the same manner, the part of the survey dealing with self-regulation is taken from literature [24]. It is in German and based on the “Academic Self-Regulation Questionnaire” (SRQ-A) [30]. The theoretical basis of the survey is the “Self-Determination Theory” (SDT) of [6]. Again, it has been previously tested and found to be reliable and valid for students from age 11 onward. The survey consists of 17 items, again using a 5-point Likert scale. The items are measuring four constructs: intrinsic regulation (IN), identified regulation (ID), introjected regulation (IJ), and external regulation (EX). These constructs can be combined to determine the self-determination index (SDI). It is calculated as $SDI = 2\cdot IN + ID + IJ - 2\cdot EX$.

Additionally, the survey, which was anonymous, asked for the gender and also asked the students to grade both the programming environment they have been using and the subject of Informatics using the German grading scheme that they are familiar with (1 to 6, 1 being the best).
4. ANALYSIS

This section presents the result of our statistical analysis of the collected data. The findings are discussed in the next section. Note that all statistical tests are two-sided whenever applicable, since we did not assume any group to perform better or worse a priori. If a one-sided test has been used, it is explicitly noted.

4.1 Pre- and Post-Test

From the Scratch class all 29 students were present for both tests. From the Karol class, only 24 students were present for both tests. Table 1 shows the results for both classes and both tests, when simply distinguishing – for each task of the tests – whether the given answer was correct, incorrect or left empty.

<table>
<thead>
<tr>
<th>Scratch Class</th>
<th>Pre-Test</th>
<th>Post-Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>correct</td>
<td>165 (63.2%)</td>
<td>218 (83.5%)</td>
</tr>
<tr>
<td>incorrect</td>
<td>78 (29.9%)</td>
<td>41 (15.7%)</td>
</tr>
<tr>
<td>left empty</td>
<td>18 (6.9%)</td>
<td>2 (0.8%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Karol Class</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>correct</td>
<td>125 (57.9%)</td>
<td>163 (75.5%)</td>
</tr>
<tr>
<td>incorrect</td>
<td>53 (24.5%)</td>
<td>45 (20.8%)</td>
</tr>
<tr>
<td>left empty</td>
<td>38 (17.6%)</td>
<td>8 (3.7%)</td>
</tr>
</tbody>
</table>

Table 1: The results of the pre- and post-tests. Given is the total number of tasks per class and test. As described above, there were 9 tasks in each test.

Next, we scored the tests. In contrast to the Bebras scoring scheme which also incorporates the difficulty, we simply scored each correct task with a 1, each incorrect task with a -1 and left out all tasks that were left blank. This was done because the original difficulty did not yield any valuable information in our case. This creates a score for each student and both tests. The minimal and maximal values that could be achieved in each test are -9 and 9 as there are 9 tasks. The mean value for both classes is 3.00 in the pre-test (with a standard deviation of 3.65 for the Scratch class and 2.95 for the Karol class). For the post-test, the mean value for the Scratch class is 6.76 and 5.08 for the Karol class (with a standard deviation of 5.02 and 5.45 respectively). When comparing the mean value of pre- and post-tests of each class using a Wilcoxon test, both differences are significant with $p < 0.004$ in both cases.

We were particularly interested in whether or not there was a significant difference in the development of these scores between the two groups. Therefore, we conducted a Mann-Whitney test on the results of the pre-test as well as on the differences between the post- and pre-test for each student. The difference can be seen as a measure of the increase in abilities. The tests reveal that in the beginning, both classes are equally “able” as there is no significant difference between the pre-test results ($p = 0.68$). The increase, however, shows a significant difference ($p = 0.047$) with the Scratch class having the higher value of the two groups.

4.2 Graded Exam

The graded exam had two tasks that were scored with a total of 14 points. When looking at the mean score-sum of the two classes, there is no significant difference. The Karol class scores slightly higher with 8.93 in contrast to 8.81 for the Scratch class (the standard deviation is 3.16 and 3.09 respectively). When looking at the single (sub-)tasks, there is a significant difference to be found between the two classes for the task of describing the effects of a given program in natural language. The Scratch class did perform better, with a score average of 2.43, compared to 1.95 for the Karol class ($p = 0.045$ for a one-sided test). For the other tasks – translating a program code into a Nassi-Schneiderman diagram and writing a program – the Karol class performed slightly better, however not significantly so.

4.3 Surveys

There were 26 responses for each class concerning the surveys. When analyzing the short-scale of intrinsic motivation, the sub scales are giving the results as shown in Table 2. The difference between the two groups is significant ($p = 0.047$) for the first sub scale (interest/enjoyment).

<table>
<thead>
<tr>
<th></th>
<th>Scratch class</th>
<th>Karol class</th>
</tr>
</thead>
<tbody>
<tr>
<td>interest/enjoyment</td>
<td>3.99 (0.93)</td>
<td>3.46 (0.93)</td>
</tr>
<tr>
<td>perceived competence</td>
<td>3.63 (0.83)</td>
<td>3.60 (0.79)</td>
</tr>
<tr>
<td>perceived choice</td>
<td>3.63 (1.07)</td>
<td>3.49 (0.63)</td>
</tr>
<tr>
<td>pressure/tension</td>
<td>2.17 (0.81)</td>
<td>2.01 (0.61)</td>
</tr>
</tbody>
</table>

Table 2: The intrinsic motivation sub scales. Given is the mean of the 5-point Likert scale. The standard deviation is given in parentheses.

The results concerning the perceived self-regulation are shown in Table 3. The self-determination index SDI can be calculated for the two classes. It is 1.99 for the Scratch class and 2.34 for the Karol class (standard deviation of 3.32 and 2.95 respectively). The only difference that is significant between the two classes is for the sub scale of identified regulation with $p < 0.05$ for the one-sided Mann-Whitney test.

Concerning the grades that the students were asked to give both to the subject and to the programming environment that they were using, the mean grade for the subject Informatics is nearly identical (2.83 and 2.81 on a scale from 1 to 6, 1 being the best grade). The mean grade for the programming environment is 2.15 for the Scratch class and 2.63 for the Karol class. This difference is “almost” significant with a one-sided test ($p = 0.059$).
of “programming in real-life” it seems to be the case that the students at their age are too young to have a real grasp – even though Scratch was more fun for the students! Since the identified regulation is higher for the Karol class. This shows a significant difference between the two classes.

Concerning the self-regulation, there is only one sub scale that shows a significant difference between the two classes. The identified regulation is higher for the Karol class. This is very interesting as it shows that the programming environment is perceived more relevant for “real-life” than Scratch – even though Scratch was more fun for the students! Since the students at their age are too young to have a real grasp of “programming in real-life” it seems to be the case that the way they are using Scratch does not relate to something they expect programmers to do – or maybe they are assuming that “playing” with Scratch is just that – playing – and therefore unrelated to “real life”.

The pre-test shows that the two classes are having a very similar level of ability before the lessons – there are no significant differences in the test results. This is to be expected, naturally, as the children typically didn’t encounter any of the relevant material before the lectures. Nevertheless, it could have been possible, as we didn’t have any influence on their prior education in grade 6. When analyzing the results of the post-test, first, it is obvious that the students of both classes gained relevant knowledge and abilities. For both classes, there is a highly significant difference between their pre- and post-test results. So, regardless of the chosen programming environment, it was possible to foster learning of the relevant concepts. As discussed above, we do not assume that using exactly the same tasks for pre- and post-test has an influence on the results in this case. Also, we do not assume that other subjects have any influence on the students’ abilities in the relevant concepts of the Informatics lessons.

While both classes did gain relevant knowledge and abilities, it is interesting to note that the Scratch class did perform significantly better in the post-test. One reason for this might be that the particular set of tasks that we chose for the test somehow presented an advantage for the children who were using Scratch. While there is no obvious reason why this should be the case, we can nevertheless not fully exclude the possibility. What does seem reasonable and what is also in accordance with theories of teaching and learning, however, is that the increased intrinsic motivation does result in better results for the Scratch class. Interestingly, the higher identified regulation of the Karol class seemingly does not compensate this effect. It may also be that the visualization of the program structure as opposed to program flow is a better learning aid for the children and therefore the Scratch class performs better.

Why then, did the Scratch class not also perform better in the graded exam? While a conclusive answer to this fact is not possible, it may be an artifact of the way the exam was designed: Since Karol is a lot less flexible in its possibilities when compared to Scratch and since our objective was to keep the exam tasks as similar as possible, the resulting tasks may have been more designed towards Karol. This, in turn, may have positively affected the Karol class in their performance, since the tasks of the exam are more akin to the exercises they were already familiar with from the lectures. For the Scratch class, the tasks may have been somewhat “new” or different from the way they were accustomed to before the exam. Also, since the possibilities of Karol are more restricted, the solution to a task often is more “obvious” due to the limited ways of controlling the robot – in comparison to Scratch. That the Scratch class did perform better in the sub-task of understanding code and describing its effects is most probably due to the clear visualization of the code structure in Scratch. Again, the visualization of program flow in Karol does not seem to bring an advantage for these kind of task. Interestingly though, the sub-task of translating a program code into a Nais-Schneiderman diagram has been solved equally well by both classes. So, the difficulty there does seem to be using the new representation and not understanding the one the chil-

<table>
<thead>
<tr>
<th></th>
<th>Scratch class</th>
<th>Karol class</th>
</tr>
</thead>
<tbody>
<tr>
<td>intrinsic regulation</td>
<td>3.08 (0.98)</td>
<td>3.07 (0.87)</td>
</tr>
<tr>
<td>identified regulation</td>
<td>2.67 (1.02)</td>
<td>3.21 (1.08)</td>
</tr>
<tr>
<td>introjected regulation</td>
<td>2.22 (0.89)</td>
<td>2.21 (0.75)</td>
</tr>
<tr>
<td>external regulation</td>
<td>2.31 (0.75)</td>
<td>2.40 (0.51)</td>
</tr>
</tbody>
</table>

Table 3: The sub scales of the self-regulation scale. Given is the mean of the 5-point Likert scale. The standard deviation is given in parentheses.

4.4 Concept Maps

In the pre-test, the students of the Scratch class drew on average 3.00 meaningfully labeled propositions, almost exactly as many as the Karol class (3.08). In the post-test, the mean of the number of such propositions grew to 3.81 for the Scratch class and 4.04 for the Karol class. Both increases are significant (with $p < 0.05$), however the difference between the two classes is not.

5. DISCUSSION

The first result that we want to discuss concerns the students’ perception of their programming environment. From the data above, it seems like Scratch is the clear winner. The Scratch class has a significantly higher value on the sub scale of interest/enjoyment regarding their intrinsic motivation. Also, they are giving a better grade to the programming environment than the Karol class. This result in itself is not very surprising, as – in contrast to Karol – Scratch is specifically designed to foster fun in programming [27]. It is interesting, however, that the grades given to the subject itself do not seem to be influenced by the programming environment, as there are no significant differences between the two classes.

Concerning the actual abilities, the picture is not as clear cut. While the Scratch class did perform better on the post-test as discussed in more detail below, there are no meaningful differences to be found in the graded exam. The better performance in the post-test is especially interesting, since there is no significant difference on the intrinsic motivation sub scale regarding the perceived competence. In other words, the students of the Scratch class did perform better on average than the students of the Karol class, but they did not perceive themselves as more competent. One explanation might be, that the survey was conducted right after the students received the results of the graded exams. Since both classes did perform equally there, it may be that the students were mostly influenced by the result of the exam when judging their own competence in the survey.

It is not surprising that the remaining two sub scales of the intrinsic motivation test do not show any significant differences between the classes. The perceived choice and the perceived pressure/tension are mostly influenced by the educational setting – which was kept nearly identical for both classes.

Concerning the self-regulation, there is only one sub scale that shows a significant difference between the two classes. The identified regulation is higher for the Karol class. This is very interesting as it shows that the programming environment is perceived more relevant for “real-life” than Scratch – even though Scratch was more fun for the students! Since the students at their age are too young to have a real grasp of “programming in real-life” it seems to be the case that
dren are accustomed to. When taking into account that the diagrams are representing the program structure in a different way than Scratch (especially visible for the conditional statement) and also there are no colors used in the diagrams, the results are even more understandable.

Since neither Scratch nor Karol are focusing on an object-oriented view, it is not surprising that there are no differences between the classes in the concept maps. The increase in complexity that has been observed nevertheless is most probably due to the curriculum, that puts a constant emphasis on object-orientation. For example the robot in Karol and the sprite in Scratch are presented as an object of an appropriate class and the program is introduced as a method that is executed for this particular object.

What has been left out of the investigation so far is the apparent higher willingness of taking a risk of the Scratch class, as shown in Table 1. This may indicate a higher self-efficacy of the students using Scratch. Similar results have been reported in [28].

6. CONCLUSION AND FUTURE WORK

The results that were discussed in the last section draw a pretty clear picture with regard to the research question underlying our experiment: Both the intrinsic motivation and the abilities – as measured by our test – have been higher/better for the Scratch class when compared to the Karol class. In a way, this only repeats a recurring theme throughout literature, even if there are exceptions, as presented above [16]. The only aspect in which Karol “won” is the identified regulation. Clearly, this is also an important aspect of a compulsory subject in secondary schools. However, due to the young age, fostering intrinsic motivation for programming probably is of higher importance than making them see a connection to programming in real-life.

It will be very interesting to see, how the professional software development will pick up on such visual programming environments in the future. Given the increasing share of touchscreen devices, tablets and so it is easy to imagine that code production will stop to rely on typing and instead may just as well happen in a visual block-oriented way similar to Scratch. Professional software development tools are also usually putting an emphasis on visualizing the program flow – by allowing a stepping through the code for debugging, for example. So, the future software development could be comprised of tools that are focusing on visualizing flow and also allowing the code to be “clicked” together. It is noteworthy, that so far there doesn’t seem to be any didactic tool that follows this approach. It would also be very interesting to see how such a tool would perform in the experimental setting presented in this paper.

Concerning our methodology, clearly our test of abilities can be improved further in many respects. Using established tasks from the Bebras contest is an obvious starting point. However, identifying whether or not these tasks are actually measuring programming abilities or identifying competencies that are related to programming are necessary further steps for similar future experiments. Also, we didn’t even use all of Scratch’s features (like the sharing of projects in the internet [27] or online forums and discussions [31]) in order to keep it more comparable to Karol’s possibilities.

Lessons that are designed for Scratch from the get-go will most probably even increase the positive effects that we identified. However, in our setting we cannot fully exclude the possibility of the teacher influencing the two classes differently (e.g. due to personal preferences).

Underlying our research question was the idea, that visualizing program flow may be an important aspect of learning algorithmic thinking and the basics of (procedural) programming. The results do not seem to indicate this. Instead, it seems to be more important to visualize the structure of the program. However, it will be interesting to see whether or not students who started with Karol will easier pick up a “real” programming language in the later grades. Typically, in grade 10 the students are learning Java. Interesting in this regard, however, is an observation made in [22]: Using Scratch does seem to foster several “bad habits” in coding – which may have an impact when switching to another programming language. In contrast, in [17], Scratch is found to have a positive impact on the switch to Java: “when asked via surveys at term’s end to reflect on how their initial experience with Scratch affected their subsequent experience with Java, most students (76%) felt that Scratch was a positive influence, particularly those without prior background”.

We are planning to investigate this in the future.

Also, there are many additional experiments that could provide further insight into the learning curve of programming. For example, it may be interesting to have two classes learn both Karol and Scratch but in opposite order. Maybe teaching both environments with their respective focus on program structure and program flow may provide a “best of both worlds” approach for the students.

In conclusion, the experiment indicates that using either programming environment shows an increase in abilities and therefore is most probably not of crucial importance for the learning outcomes. However, if given a choice, it seems favorable to choose an environment that visualizes program structure and fosters intrinsic motivation for programming in the students.

7. REFERENCES


