Designing and comparing two Scratch-based teaching approaches for students aged 10-12 years – extended version

Nienke van Es

Johan Jeuring

Technical Report UU-CS-2017-015
October 2017

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Designing and comparing two Scratch-based teaching approaches for students aged 10-12 years

Nienke van Es
Utrecht University
nfjvanes@gmail.com

Johan Jeuring
Utrecht University and Open University of the Netherlands
j.t.jeuring@uu.nl

ABSTRACT

Programming and computational thinking are becoming more important in primary education. This raises the question of how different approaches to teaching programming in primary schools compare with each other. We designed two approaches to teach programming to primary school students. One approach uses the instructionistic 4C/ID model, the other approach uses constructionism. The learning gains of these two approaches were compared using a pre- and post test. In total, 129 students from two different schools participated. A significant difference (p = .037, d = .59) between the two approaches was found on one of the schools, favoring the 4C/ID approach. On the other school and for the total group no significant difference was found. This is explained by the different backgrounds between the students from the different schools.

This paper is an extended version of a paper that is published in Koli Calling 2017, and contains more details about the research performed.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; **K-12 education**;

KEYWORDS

teaching Scratch programming, primary school, comparing teaching approaches, constructionism, 4C/ID

ACM Reference format:
Nienke van Es and Johan Jeuring. 2017. Designing and comparing two Scratch-based teaching approaches for students aged 10-12 years. In *Proceedings of Koli Calling 2017, Koli, Finland, November 16–19, 2017,* 9 pages.
DOI: 10.1145/3141880.3141883

1 INTRODUCTION

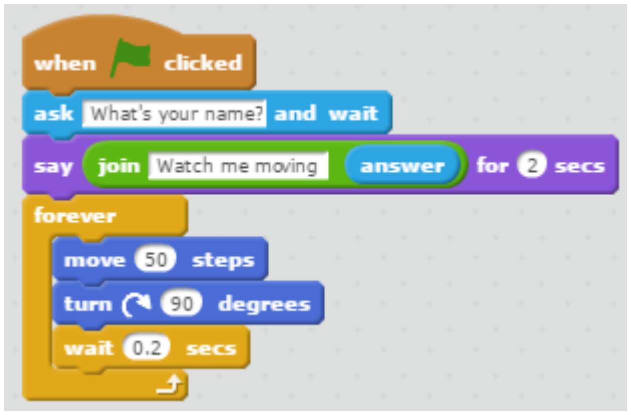
We live in a world where a child spends a couple of hours per day interacting with software on various kinds of computers. To understand this world, children need to get an idea of what is behind software, and how software is constructed. For young students, learning how to develop a computer program has more advantages than just gaining the ability to develop a program. The process of learning to program also improves problem solving, logical thinking, and organizational skills (Lee, 2011). These skills are also applicable to real world problems and other courses. In our own country, The Netherlands, the parliament has asked the secretary of education to find a place for programming in primary education.

After finding a place for programming in the curriculum, the immediate follow-up question is how to teach programming effectively. Quantitative research on learning gains of different programming teaching approaches is sparse (Jeuring, Corbalan, van Es, Leeuwestein, & van Montfort, 2016). Many teachers don't feel confident about how to teach programming, and are experimenting with various ways to approach it. Teaching programming needs to be approached carefully: confused teachers can harm young students' attitude towards programming (Duncan, Bell, and Tanimoto, 2014). It is important to provide teachers with successful approaches to teach young students programming. This paper contributes to this goal by describing an experiment in which we design and compare two different approaches to teaching programming.

The two teaching approaches we investigate are based on constructivism and the 4C/ID model, respectively. Using constructionism in teaching programming is very popular (Baytack, & Land, 2011; Bruckman, & De Bonte, 1997; Dasgupta et al., 2016; Ngai, Chan, Leong, & Ng, 2013). With this approach students build their own meaningful product. The 4C/ID model is designed to teach complex cognitive skills. Complex cognitive skills are described as skills that are time consuming to acquire (Van Merriënboer, & Dijkstra, 1997). As an example of a complex cognitive skill, Van Merriënboer and Dijkstra (1997) mention computer programming. Using the 4C/ID approach, teachers often explain a topic using frontal instruction. After the instruction, a student practices the skills with worksheets.

We design series of programming lessons based on the above two teaching approaches, and perform these lessons in five different classes at two different primary schools. At each school we perform both teaching approaches. Students are in between 10 and 12 years old. We compare the learning gains of the two teaching approaches to find out how the approaches affect the development of programming competencies of students. The aim of this comparative design-based research is to determine which of the teaching approaches results in higher learning gains.

We use Scratch to teach programming at primary school. This tool is often used to study teaching approaches and to teach students programming (Aritajati, Rosson, Pena, Cinque, & Segura, 2015; Dasgupta, Hale, Monroy-Hernández & Hill, 2016; Flannery et al., 2013; Franklin et al., 2015). According to the Scratch website, it is used in more than 150 countries and has more than nineteen million shared projects (http://scratch.mit.edu). There are a number of reasons why we choose to use Scratch for teaching programming



```
when [flag] clicked
ask [What's your name?] and wait
say (join [Watch me moving] (answer)) for (2) secs
forever
    move (50) steps
    turn ↻ (90) degrees
    wait (0.2) secs
```

Figure 1: Scratch example code: the cat asks for your name, says something using your name, and then walks a square.

at primary school: Scratch is designed for novice programmers without any programming experience (Aritajati et al., 2015; Lee, 2011), and the interface of Scratch is student-friendly (Franklin et al., 2015). Scratch has a drag and drop interface that provides visual support using programming blocks. Two blocks only fit together when the corresponding code is syntactically correct. A programmer uses the blocks to write software or games without writing the textual code (Lee, 2011). Figure 1 gives an example of Scratch code.

This paper is an extended version of a paper that is published in Koli Calling 2017, and contains more details about the research performed.

This paper is organised as follows. Section 2 further introduces the two teaching approached we use. Section 3 discusses the design of our research, and Section 4 the research method. Section 5 gives the results, which are discussed in Section 6. Section 7 concludes.

2 BACKGROUND

We use two teaching approaches in this study: constructionism and 4C/ID. This section discusses the learning theory behind and the practical implications of each of the approaches. We explain where the two teaching approaches, as we develop them, are similar and how they differ from each other. The outcomes of this research not only depend on these theories, but also on how well we manage to use the characteristics of the theories in the development of education (DiSessa, & Cobb, 2004).

2.1 Constructionism

Constructionism is a learning theory that builds on Piaget's constructivism. In constructionism, the learner actively constructs knowledge. The theory emphasizes that a learner constructs this knowledge through developing, for example, a computer program, in a development cycle in which sharing plays an important role (Kafai & Resnick, 1996, pp. 176-178). The product to be developed should be meaningful for the student, because constructing personally meaningful products is effective for learning. The approach assumes a teacher to act as a coach instead of a facilitator of knowledge (Kafai & Resnick, 1996, pp. 176-178, 208-214).

Papert and Harel (1991) describe constructionism as "learning-by-making", and mention "playful" as an important characteristic. In their book, Kafai and Resnick (1996) characterise constructionism by learning through design, and learning in communities. Ngai and colleagues (2013) characterise constructionism as a way to incrementally enlarge and build knowledge into fundemental concepts. Bruckman and De Bonte (1997) argue that the process of learning is successful when the students are working on projects that are challenging their skills and give opportunities for creativity. Another important aspect described by Bruckman and De Bonte is a community in which students can help each other. Within the learning process, the students can either ask their teacher or fellow students for help and construct knowledge together. This corresponds to learning in communities as described by Kafai and Resnick (1996). An example of using Scratch with the constructionist approach is to let students design and program their own games.

2.2 The 4C/ID model

The four component instruction model, 4C/ID in short, is a guideline for instructional design. The model consists of four components which, when combined, support complex learning. The four components are learning tasks, supportive information, just in time information, and subtask practice. Whereas other design models follow the theory that a complex learning task can be achieved by the sum of the parts, this does not hold for the 4C/ID model. The 4C/ID model is based on the theory that the whole is more than the sum of the parts. There is evidence that the theory in which the whole is equal to the sum of the parts is not valid (Van Merriënboer, Clark & De Croock, 2002).

One of the major assumptions of the 4C/ID model is that complex skills can only be learned by doing. The learner must be confronted with tasks in which (s)he can perform the skills that have to be learned (Van Merriënboer & Dijkstra, 1997). With learning by doing, a student will learn procedural knowledge. According to Van Merriënboer and Dijkstra, the 4C/ID model makes a distinction between controlled and automatic information processing for procedural knowledge. Controlled information processing requires effort from the learner, whereas automatic information processing requires hardly any effort because it just happens. It is therefore important to determine the desired behaviour of the learner. With the knowledge of what behaviour can be expected, a test can determine whether a learner managed to learn the skill. More importantly, the teacher can determine if the skill is automated or controlled (Van Merriënboer & Dijkstra, 1997).

Rule automation helps in achieving automated processing. Rule automation is the construction of specific procedures that can be used to decompose problems into subproblems. The procedures can then be used to solve these subproblems. For effective controlled processing, schema acquisition is important. A schema provides knowledge that can be used to solve particular problems. If a schema is more developed it can provide more generalized knowledge. Rule automation and schema acquisition both occur when learning a cognitive complex skill. With this knowledge, the four components of the 4C/ID model are developed (Van Merriënboer & Dijkstra, 1997).

Learning tasks, supportive information, just in time information, and subtask practice are the four components of the 4C/ID model. Together they provide a basis that can be used in different contexts. The first component, learning tasks, is based on the principle that learners learn by performing meaningful tasks. By performing these tasks the learner integrates knowledge and subskills in her existing knowledge. All subskills are practiced in the context of a whole complex task. The second component, supportive information, helps in obtaining the knowledge necessary for the learning tasks. The just in time information consists of procedural knowledge needed to perform the task. This information is given just before the learner needs it. The last component is subtask practice, which means that a learner practices subtasks as long as these are not automated (Van Merriënboer et al., 2002; Poortman & Sloep, 2006).

An example of a learning task for programming in Scratch is: let the avatar ask your name and then greet you with your name. The supportive information for this task is about variables. The just in time or procedural knowledge for this task is information about how to create an avatar and use blocks and variables in Scratch. Some subtasks to practice are creating variables and changing or combining them with the help of blocks.

2.3 Research question

In previous research, Baytak, and Land (2011) used the constructionistic theory to develop a Scratch-based teaching approach for 5th grade students. In this approach, students are asked to design an educational game for 2nd graders in Scratch. The students develop the games in 21 sessions of 45 minutes. From interviews, transcriptions, and the collected games, the researchers concluded that the students were motivated to make a game, and that all ten students succeeded in making a game. A limitation of this study is the absence of pre- and post assessments to identify gains in programming skills.

Kalelioglu and Gülbahar (2014) use frontal instruction to teach students programming with Scratch. They start simple and increase the complexity of the exercises every week. The lesson series consists of five lessons of one hour. The aim of the research is to find a difference in the students' perceptions of their problem solving skills. Learning gains of the approach are not investigated. The main conclusion is that there is no significant difference in the students' perceptions of their own problem solving skills.

None of the above studies aims to identify gains in programming competencies. We think that with the right assessment, these teaching approaches are suitable to test learning gains. We design two lesson series for teaching programming in Scratch: a lesson series based on the constructionistic approach of Baytak and Land's (2011) study, and a lesson series based on the 4C/ID model. The aim of our research is to find out which approach leads to higher learning gains in programming competencies of students: a constructionistic approach or the 4C/ID approach. Our main research question is:

> *Which teaching approach, constructionism or 4C/ID, results in higher learning gains in terms of programming competencies for students aged 10-12 years?*

2.3.1 Hypothesis. Sawyer (2005) argues that the instructionistic approach mostly teaches facts and procedures to students. In todayâĂŹs society students need more than just facts and procedures, namely a deeper understanding of complex concepts. Students also need the skills to use this understanding to create new ideas, knowledge, products, or procedures. Instructionism results in knowledge which is hard to use outside the classroom, because the student has to know how the learned knowledge is applicable in new situations (Sawyer, 2005, pp.1-3).

On the other hand, Kirschner, Sweller, and Clark (2006) argue that guidance and instruction are more effective than constructivistic approaches. They also argue that less guidance may cause misconceptions. The arguments are based on the knowledge of the human cognitive architecture. The main components of this architecture are the working memory, the long term memory, and the relation between these two kinds of memory. Learning something implies a change in the long term memory. When new knowledge is provided, the capacity of the working memory and the duration of how long the working memory can hold the information are two limiting factors. These limitations are lifted when the memory can work with familiar knowledge. Minimal guiding approaches, like problem based learning, are causing an overloading of the working memory and result in less effective learning.

Brunstein, Betts, and Anderson (2009) do not agree with the above arguments. According to them, there is a difference between no or very little guidance and helping students by giving hints and putting them on the right track when they are developing their products. When there is guidance during the learning process, a minimal guiding approach can be effective.

There is no clear answer to the question which approach results in higher learning gains. Therefore, our hypothesis is that there is no significant difference in learning gains.

3 DESIGN

We develop five lessons for each teaching approach. For the 4C/ID model, we used the ten steps to complex learning as a guideline to develop these lessons. These ten steps are based on the 4C/ID model and the Instructional Systems Design (ISD) process (Van Merriënboer, & Kirschner, 2012). Each lesson covers one of the main programming themes from the Dutch computer science curriculum (Barendsen, & Tolboom, 2016): sequences, conditionals (if-statements), loops, and combined conditions and variables.

Each lesson starts with a short introduction into the theme by working through some case studies. We choose this explanatory inductive strategy because it works well for students with little prior knowledge and is time effective (Van Merriënboer et al., 2002). After the introduction, the students work individually on an assignment distributed through hand-outs. The hand-outs incorporate the other components of the 4C/ID model. The just-in-time information is presented in red outlined squares. These squares give step by step instructions on how to deal with the practical aspects of Scratch. The part time practice assignments are presented in purple outlined squares. These squares contain small exercises to get more familiar with important subskills.

In the constructionist approach, the students develop their own game. To ensure that students have some prior knowledge before developing a game, the first lesson is spent on the basics of Scratch and programming. The students explore the programming blocks by using the Scratch cards from the Scratch website, which encourage

4C/ID	Constructionism
Instruction at the start of each class about the main concept	No instruction, except for explaining the assignment
Information on how to use Scratch and how to program	Students explore Scratch and programming by themselves with a little help from the Scratch cards
Small assignments to help with the bigger assignment Assignments to illustrate a single concept	One assignment to learn all concepts

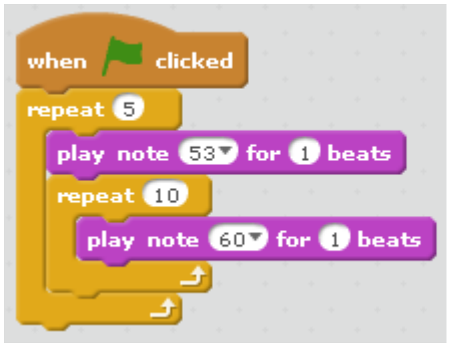
Table 1: Differences between the two teaching approaches

the students to learn by exploring (Wilson, Hainey, & Connolly, 2012). After this first lesson, the students start with developing their own game. They get three and a half hour to complete a game. In the last half hour students play the games developed by other students, and ask questions about them. The main sources for programming knowledge are the Scratch cards and the help they request from the researcher.

While guiding the students, the researcher tries to stimulate cooperation between classmates by passing questions to students who asked the same thing. This kind of cooperation is strongly suggested for constructionism (Sawyer, 2005, pp. 39-40; Baytak, & Land, 2011). When helping the students, it is important to tell the students just enough so they can go on by themselves. It is not the intention to program the game for the students (Blaho, & Salanci, 2011). Students who do not know how to start receive a design form. This form contains some questions to guide the students in the process of designing a game. For example, the students are asked to make a sketch of the layout of the game, and to write down some rules.

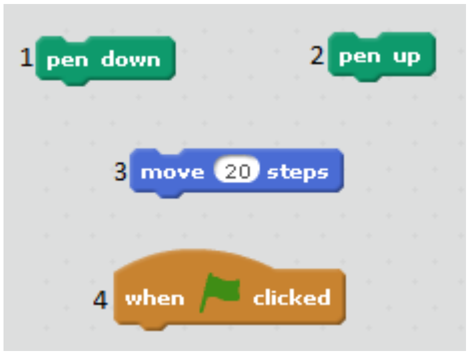
The main differences between the two approaches are summarized in Table 1. There are also some similarities between the approaches. Students are working on real world tasks in both approaches. There is some guidance at the start of the game design assignment for students who need this. The students who need help at the start are guided using a top-down approach, while the other students have the option to work bottom-up.

To test if there is a learning gain after performing the five lessons, we develop a pre- and post test. The two tests cover the three main concepts we test, namely sequences, conditions, and loops. These are derived from the renewed Dutch curriculum for computer science for secondary schools. This curriculum mentions sequencing, repetition (or loops), conditions, and variables as basic programming constructions (Barendsen, & Tolboom, 2016. pp. 16-17). Barendsen and Tolboom also mention some constructions to support abstraction by programming, but we feel that these topics are less suitable for primary school students. The tests do not contain specific questions about variables, because this topic is implicitly covered in the other questions. The test groups the questions by topic. For the post test we use the same kind of questions as for the



```
when [flag] clicked
repeat 5
    play note 53 for 1 beats
    repeat 10
        play note 60 for 1 beats
```

Figure 2: Question inspired by Lewis (2010)



```
1 pen down
2 pen up
3 move 20 steps
4 when [flag] clicked
```

Figure 3: Question where a student has order programming blocks correctly

pre test, with a slightly changed context. We also change the order of the questions in the post test.

The literature says little about what kind of questions related to programming this age group can deal with. Lewis (2010) gives some questions in her test for the same target audience. We use these examples to develop two similar questions in our tests. Figure 2 shows an example question from the pre test. Students are asked how many beats the program lasts in total, and how many beats note 60 is played.

The other questions are based on the description of learning goals in the Dutch curriculum for computer science for secondary education, namely developing code, adapting code, and explaining code (Barendsen, & Tolboom, 2016, pp. 36).

One part of the questions is about sequencing. These questions ask to put the programming blocks in the right order. Such a question tests the ability of a student to develop a program. In some of the questions, we ask students to give the right order if something in the program changes. Such a question is derived from the learning goal that students are able to adapt a program when the requirements change (Barendsen, & Tolboom, 2016, pp. 36). Figure 3 shows a part of a question in which programming blocks have to be put in the right order. Students have to order the blocks to obtain a program that draws a line of 20 steps.

The last kind of questions are the questions where students explain a piece of code. We use a question as shown in Figure 4 to

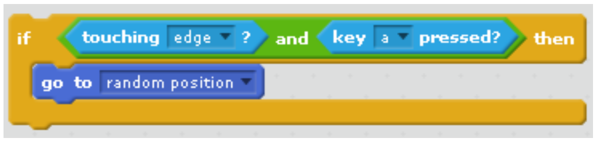


Figure 4: Question where students were asked to explain what happened

test students on explaining the structure and working of a program (Barendsen, & Tolboom, 2016, pp. 36). Such questions are similar to some of the review questions in Calder's book (2011), in which students have to explain what a program does when it is executed.

4 METHOD

We perform a comparative design-based study. To answer the research question, we design two sets of lessons, which we teach at two schools. The two lesson sets are based on constructionism and the 4C/ID model, respectively.

The first author taught the Scratch programming lessons on one school for one hour per week. She organised one hour sessions twice a week for three weeks at the other school. The experiment took three to five weeks, and took place in October and November 2016.

4.1 Participants

The lessons were taught at two Dutch primary schools with 5th and 6th grade students, aged nine to twelve years. The schools where the experiments took place are located in Utrecht and Wageningen. The schools participated voluntarily. We solicited schools through a newsletter for elementary schools, and got responses from the two schools at which we performed the experiments.

The students from the school in Utrecht were distributed over three classes, one 5th grade, one 6th grade and one combined 5th and 6th grade class. In total 87 students between 9 and 12 years old participated. The 5th grade students (18 female, 13 male) were on average 10,1 years old in the range from 9 to 11. Eight out of thirty-one students were familiar with Scratch. The 6th grade students (11 female, 15 male) were on average 11,1 years old in the range from 10 to 12. In this class, twelve students were familiar with Scratch. The combined 5th and 6th grade class students (14 female, 15 male) were on average 10,4 years old in the range from 9 to 12. Out of the 29 students, 10 students said they knew Scratch. The school is located in the Vogelenbuurt just outside the centre of Utrecht. The school is a public school, and is accessible for students with all cultural backgrounds and religions.

The students from the school in Wageningen were distributed over two combined 5th and 6th grade classes. In total, this school had 57 students between 9 and 12 years old that participated. The first class had 28 students (17 female, 11 male) with an average age of 10,5 years in the range from 9 to 12. In this class, only five students knew Scratch before the project started. The second class contained 29 students (13 female, 16 male) with an average age of 10,6 in the range from 9 to 12. Eleven students were familiar with Scratch at the start of the project. The school is a Catholic school.

To prevent differences between students of the different schools from influencing the research as much as possible, we taught both teaching approaches on both schools. The distribution of students over the teaching approaches is shown in Tables 2 and 3. We assumed that similar groups within the schools were equally diverse. This means that we assumed that the different groups within the schools had an equal spread of ethnic backgrounds. Also, we assumed the students had approximately the same learning history, and the same spread of learning levels. So we assumed that by using the fixed groups in the schools, these groups were comparable within the schools.

	Constructionism	4C/ID
Grade 5	31 students	
Grade 6	27 students	
Combined grade 5-6		29 students

Table 2: Distribution of the approaches in Utrecht

	Constructionism	4C/ID
Combined grade 5-6a	29 students	
Combined grade 5-6b		28 students

Table 3: Distribution of the approaches in Wageningen

4.2 Instruments

As explained above, the two teaching approaches which were used differ in how students learned to program in Scratch. Due to the lack of computers in Wageningen, the students learned to program with an app called Pyonkee. This Scratch-based application works on iPads and contains the same programming blocks as Scratch. Pyonkee is based on version 1.4 of Scratch for iPads (https://wiki.scratch.mit.edu/wiki/Pyonkee). The worksheets and tests were not changed for these students. The explanation was given in Scratch.

To determine the internal consistency of the tests, Cronbach's alpha (Field, 2009, pp. 673-676) was calculated. This value has to be above 0.7 for the test to be accepted as reliable. Other test quality measures are the *p* value, rit, and rir of the questions. The *p* value says something about how well students answered a question, and is preferably in between 0.3 and 0.8. The rit and rir give an impression of the distinctiveness of the question, and are computed to relate the score of the single question to the total score. For the rir, in contrary to the rit, the total score minus the score of the question for which the rir is calculated is used. The rit and rit are considered good from 0.35 and up. For the pre test, Cronbach's alpha was 0.83. The other quality measures are presented in Table 4. The Cronbach's alpha of the post test was 0.81, and Table 5 presents its other quality measures.

All tests were graded by the first author. A randomly picked sample of 30 of the 129 tests, both pre- and post tests, was independently graded by a second grader. The first author determined if there were major differences in scores. For both the pre- and post

	p	variance	rit	rir
1	.44	.81	.54	.43
2a	.14	.39	.54	.47
2b	.13	.41	.52	.44
3	.39	.74	.72	.64
4	.40	.91	.65	.54
5	.60	1.12	.63	.51
6	.31	1.73	.72	.59
7	.35	.91	.62	.51
8	.27	2.30	.72	.56
9a	.20	.49	.59	.51
9b	.27	.20	.51	.46

Table 4: Statistics of the pre test

	p	variance	rit	rir
1	.62	.66	.55	.454
2a	.50	.56	.63	.55
2b	.87	.13	.47	.42
3	.50	.75	.56	.45
4	.37	1.05	.51	.38
5	.74	.75	.64	.55
6	.52	3.00	.75	.59
7a	.22	.55	.57	.48
7b	.18	.54	.62	.54
8	.64	1.73	.69	.56
9	.55	.77	.56	.46

Table 5: Statistics of the post test

test the first author discussed with the second grader the differences in how questions were scored. After this, the first author adapted the correction model where needed and corrected all the pre- and post tests again. Then, Cohen's Kappa for inter-rater reliability was calculated. The value of Cohen's Kappa has to be above 0.6 to be acceptable (McHugh, 2012). For the pre test, Cohen's Kappa was 0.79 and for the post test 0.68. Therefore, we assume there was enough agreement between the graders.

4.3 Data collection and analysis

Pre- and post test data on programming competencies of all participants was collected and graded. Data from students who did not attend the pre- or post test was removed from the final results. In total, we have data of 129 students. We used the score on the test expressed as a percentage (100%: fully correct, 0%: completely wrong) to analyse the data. Unless indicated otherwise, the analysis was performed with SPSS.

First, we tested whether the pre- and post test had a significant difference for each teaching approach and school. We performed a paired sample t-test (Field, 2009, pp. 326-330). To perform a paired samples t-test, the learning gains data should be normally distributed. If the data is not normally distributed, we have to use a similar test, without a normally distributed data condition. Such tests are called non-parametric tests. Non-parametric tests have a disadvantage compared with a t-test: they cannot use the information of the underlying distribution of the data, and are therefore less informative. To test for normal distribution of data, we used a Shapiro Wilk normality test (Field, 2009, pp. 144-148).

	Abbreviation	Shapiro-Wilk
Wageningen 4C/ID	W4	p = .584
Wageningen constructionism	WC	p = .020
Utrecht 4C/ID 5th grade	U5	p = .252
Utrecht 4C/ID 6th grade	U6	p = .888
Utrecht constructionism	UC	p = .470

Table 6: Results of the Shapiro-Wilk normality test of the learning gains

The results of the normality tests can be found in Table 6. For the Wageningen constructionism group, the p value is smaller than 0.05 which means the data is not normally distributed according to this test. However, when we look at the histogram and the Q-Q plot of the learning gains, there is no reason to assume the data is not normally distributed. Therefore, we did not perform non-parametric tests. All other groups have p values that are larger than 0.05 which implies that the data is normally distributed, and hence that the t-test can be performed. When there is a significant difference between the pre- and post test, Cohen's d will be calculated to measure the effect of the intervention.

To compare the two teaching approaches per school, we calculated the learning gain by subtracting the pre test score from the post test score. We used a two tailed t-test to determine if there is a significant difference between the two teaching approaches (Field, 2009, pp. 334-339). If there is a significant difference, we use the means of the scores to determine which teaching approach was more successful on that school. We also tested if there is a difference between the schools. Therefore, we used a two tailed t-test for both approaches on both schools. Again, if there is a significant difference, the means are used to determine which approach on what school had the higher learning gain.

We also tested if there was a significant difference between the two teaching approach for all the participants together. We used a two tailed t-test to determine if there is a difference. We also performed a two tailed t-test on some variances of the data. For any significant difference, the means will be used to determine which approach had the higher learning gains.

Lastly, we measured the experiences of the students with the programming lessons. These questions were on the front page of the post test, and consisted of seven questions. Six of these questions should be answered on a five point Likert scale. The last question asked how many hours the students worked with Scratch beside the lessons during the intervention. All the questions gave an overall impression on how students experienced the programming lessons.

5 RESULTS

This section presents the main results. First, we want to determine if there is a learning gain for the students after taking the five lessons. Table 7 shows the results of the paired sample t-test. All classes significantly improved between the pre- and post test (p <.05). The

		W4	WC	U5	U6	UC
Pre	Mean	32.23	30.96	36.44	23.64	37.93
	SD	23.69	24.69	24.15	27.61	25.39
Post	Mean	62.77	50.52	46.44	42.18	50.43
	SD	18.65	29.86	21.40	26.44	24.59
	t	7.581(25)	6.129(26)	2.402(26)	4.284(21)	4.174(27)
	p	<.001	<.001	= .024	<.001	<.001
	d	1.41	0.71	0.44	0.69	0.50

Table 7: Results of the paired sample *t*-tests for the pre- and post test for each class

		Wageningen	Utrecht	All*
4C/ID	**Mean**	30.54	13.84	19.63
	SD	20.54	21.27	22.36
	N	26	49	75
Constructionism	**Mean**	19.56	12.50	15.96
	SD	16.58	15.85	16.45
	N	27	28	55
	t	2.146(51)	.289(75)	1.028(128)
	p	.037	.773	.284
	d	0.59	0.07	0.18

Table 8: Results of comparing the two teaching approaches
*** equal variances not assumed (F = 5.782, *p* = .018)**

effect sizes (Cohen's *d*) of the school in Wageningen are higher than in Utrecht. The effect measured for the 5th grade students in Utrecht is the lowest effect of all classes. The other classes in Utrecht are in between the lowest and highest effect sizes.

The results of the *t*-test are presented in Table 8. There is a significant difference between the learning gains for the two teaching approaches in Wageningen (*p* = .037). By looking at the means, the difference is in favour of the 4C/ID approach (M = 30.54) There is no significant difference between the learning gains of the two teaching approaches for Utrecht or the schools combined (*p* >.05). However, when we take a closer look at the data we find that students scored lower on one question in the post test in comparison with the pre test. This question was about conditionals. When designing the post test we assumed the students would learn something about coordinates in both approaches and would therefore be able to answer the question. In the pre test we assumed the students did not know this and we therefore used more readable programming blocks for this question. Since most students failed to answer the question in the post test, we excluded the question from both the pre- and post test and performed the *t*-test once more. The results of these tests are presented in Table 9.

There is no difference in the main results with the excluded question. For the school is Wageningen, there is still a significant difference (*p* = .020) between the two teaching approaches. For the other groups, there is no significant difference.

Due to the small effect of the 5th grade in Utrecht, we were curious what would happen if we excluded this class from our original results. The results of this exclusion can be found in Table 10. The main difference is that there is a significant difference between the two teaching approaches for the two schools combined

		Wageningen	Utrecht	All*
4C/ID	**Mean**	38.46	18.65	25.52
	SD	22.39	23.00	24.55
	N	26	49	75
Constructionism	**Mean**	25.25	17.69	21.41
	SD	17.25	16.52	17.16
	N	27	28	55
	t	2.411(51)	.132(75)	1.123(128)
	p	.020	.848	.263
	d	0.66	0.05	.019
	Shapiro-Wilk	*p* = .140	*p* = .403	*p* = .156

Table 9: Results of comparing the two teaching approaches without the excluded question
*** equal variances not assumed (F = 9.679, *p* = .002)**

		Utrecht	All*
4C/ID	**Mean**	18.55	25.04
	SD	20.30	21.10
	N	22	48
Constructionism	**Mean**	12.50	15.96
	SD	15.85	16.45
	N	28	55
	t	*t*(48) = 1.183	*t*(88) = 2.410
	p	*p* = .243	*p* = .018
	d	0.34	0.48
	Shapiro-Wilk	*p* = .818	*p* = .142

Table 10: Results of comparing the two teaching approaches without the 5th grade of Utrecht
*** equal variances not assumed (F = 5.095, *p* = .026)**

(*p* = .018). However, there is still no significant difference for the school in Utrecht.

Finally, Table 11 presents the results of the survey on how students experienced the programming lessons. Overall, the students gave the programming lessons a score of around 4 out of 5. They scored the difficulty of developing programs in between 2.80 and 3.20. The students also scored how much they thought they had learned. These scores are between 3.40 and 3.90.

6 DISCUSSION

This study investigates the research question

> *Which teaching approach, constructionism or 4C/ID, results in higher learning gains in terms of programming competencies for students aged 10-12 years?*

Because there was not enough evidence to support one theory over the other, our hypothesis is that there is no significant difference between the learning gains of the two approaches.

The results show a significant learning gain of the students between the pre- and post test. The effect in Wageningen is larger than in Utrecht. In Wageningen, the effect of the 4C/ID approach is almost 2 times larger than the effect of the constructionistic approach. In Utrecht the effect between pre- and post test for both teaching approaches are smaller. For the 4C/ID approach, the 5th

(1) Did you like constructing programs on a computer?
(2) Did you like working with Scratch?
(3) Was it hard to develop programs on the computer?
(4) Was it hard to work with Scratch?
(5) Did you like the programming classes?
(6) How much do you think you have learned?
(7) Hours spent on programming outside classes

	(1)	(2)	(3)	(4)	(5)	(6)	(7)
W4	4.19	4	3.19	2.88	4.04	3.73	1
WC	4.04	3.74	2.94	2.89	3.96	3.41	1.06
U5	4.46	4.46	3.11	2.64	4.32	3.86	1.07
U6	4.23	4.32	2.84	2.52	3.98	3.80	5.18*
UC	4.38	4.14	2.96	2.88	4.32	3.79	1.75

Table 11: Results of the survey. For questions starting with "Did you like", 1 denoted "did not enjoy", and 5 "liked it a lot". For questions starting with "Was it hard", 1 was "very easy", and 5 "very hard". For the question about learning, 1 denoted "nothing", and 5 "a lot".
*** One of the students entered a very high amount of hours, namely 105**

grade had a small effect and the 6th grade a medium effect. The constructionistic approach in Utrecht also had a medium effect.

The *t*-test shows that there is a significant difference between the two approaches in Wageningen. The effect of this difference is a medium effect ($d = 0.59$). This difference is not measured for Utrecht or for all the students combined. The effects for these comparisons are not even small, which raises some questions about why this difference is only measured for Wageningen. Also remarkable is that the average learning gains in Wageningen (4C/ID: M=30.54, constructionism: M=19.56) are much higher than in Utrecht (4C/ID: M=13.84, constructionism: M=12.50).

6.1 Only one significant difference?

We first discuss some of the circumstances that might have caused some differences between the schools. The first difference is that the Utrecht school did not book additional time for the post test. The Wageningen school booked an extra slot for the post test. This means that the Utrecht students worked on the post test starting half an hour into the final lesson. The Utrecht students thus had thirty minutes less to learn programming. Also the transition between working on computers to focussing on a test may have influenced the results.

The differences between Wageningen and Utrecht might also be explained by the planning of the lessons. The students in Wageningen had programming lessons twice a week for three weeks. In Utrecht the students had one programming lesson for five consecutive weeks. The programming lessons in Wageningen were closer to each other, which might have helped the students remember prior knowledge. By remembering prior knowledge, the working memory can more efficiently process new information (Kirschner et al., 2006). This may have created an advantage for the Wageningen students.

Another explanation for why the difference is only present in Wageningen might be that the schools have slightly different approaches to learning. The Utrecht school propagates a learning by doing approach, and mentioned so at the intake discussion. It might hence be the case that the Utrecht students are more used to constructionistic approaches than the students in Wageningen. When asked, the Wageningen school said that their students were also familiar with learning by doing approaches to learning. However, the students in Wageningen could have had a preference for the 4C/ID approach.

Sun, Pan, and Wang (2010) argue that researchers always have to put the effect sizes in context of the subject and research field. If we take all above mentioned considerations into account, the planning of the lessons and post test, we can still say that the effect between pre- and post test for the 4C/ID approach in Wageningen is large. Comparing with the other effect sizes between pre- and post test, the 4C/ID approach in Wageningen is two times larger than the highest effect size of the others. The difference between the 4C/ID and constructionistic approach in Wageningen had an effect of 0.59. Because there was no effect between the two approaches in Utrecht and with the two schools combined, the effect size from Wageningen also indicates that the 4C/ID approach was more successful in Wageningen.

The main learning theory underpinning the 4C/ID approach is the human cognitive architecture (Kirschner et al., 2006). The students get the information in small pieces so the working memory can process them using prior knowledge. This will eventually result in a change in the long term memory, which is needed to develop learning schemes (Van Merriënboer, & Dijkstra, 1997). The Wageningen students had less time between programming classes. Students might have had an advantage here, because they remembered the prior knowledge better.

6.2 Further analysis

After looking at the overall data, we also looked at two special sets of data, obtained through filtering the original data. The first set was obtained by excluding a question the validity of which was doubtful. Using this data, the effect size in Wageningen between 4C/ID and constructionism slightly rose to 0.66. In Utrecht and for the two schools combined, there was no difference.

The next data set was obtained by excluding the data from the 5th grade in Utrecht. This was the youngest group that depended most on help when working on the exercises, and had the smallest learning effect. Excluding this class from the data gives a significant difference between the teaching approaches for the total group, with an effect size of 0.48. This difference did not occur in Utrecht alone.

Using these filtered data sets, there is no proof that there is a significant difference between the two approaches for the Utrecht school. The question of doubtful validity had little effect on this school. Also the 5th grade students did not cause noise for Utrecht. For the total group, excluding the 5th grade from Utrecht, there was a significant difference with a small effect. We think this difference is mostly explained by the effect of Wageningen on the total group.

7 CONCLUSIONS

The 4C/ID approach leads to significantly better learning gains when learning programming than a constructionistic approach for the Wageningen school. We thus reject our hypothesis for the Wageningen school. However, we cannot reject our hypothesis for the Utrecht school or the total group. The difference between the schools might be explained by students having a preference for one teaching approach over another, and the students from the different schools have different backgrounds. All students seemed to like the programming lessons, no matter what approach was used.

For future research it would be interesting to see what happens if the lessons are performed with the same time between the lessons on both schools. Additionally, an adapted post test in which the question of which the validity is dubious is excluded should be used. It would also be interesting to see what happens when the research is performed with more homogeneous classes and more students. For example, all 5th and 6th grade classes or all combined 5th and 6th grade classes. With the separate 5th and 6th grade classes there might be a difference in age groups.

REFERENCES

Aritajati, C., Rosson, M. B., Pena, J., Cinque, D., & Segura, A. (2015). A socio-cognitive analysis of summer camp outcomes and experiences. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 581-586.

Barendsen, E., & Tolboom, J. (2016). Advies examensprogramma informatie havo/vwo. From: http://www.slo.nl/organisatie/recentepublicaties/adviesinformatica/, Enschede: SLO.

Baytak, A., & Land, S. M. (2011). An investigation of the artifacts and process of constructing computers games about environmental science in a fifth grade classroom. *Educational Technology Research and Development, 59(6)*, 765-782.

Blaho, A., & Salanci, L. U. (2011). Informatics in primary school: principles and experience. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*, Springer Berlin Heidelberg, 129-142.

Bruckman, A., & De Bonte, A. (1997). MOOSE goes to school: A comparison of three classrooms using a CSCL environment. In *Proceedings of the 2nd international conference on Computer support for collaborative learning, International Society of the Learning Sciences*, 20-27.

Brunstein, A., Betts, S., & Anderson, J. R. (2009). Practice enables successful learning under minimal guidance. *Journal of Educational Psychology, 101(4)*, 790-802.

Calder, T. (2011). Learning to Scrath a beginners guide to computer programming for kids. Prince Rupert, ISBN: 978-0-9811587-1-6.

Cohen, J. (1992). A power primer. *Psychological bulletin, 112(1)*, 155.

Dasgupta, S., Hale, W., Monroy-Hernández, A., & Hill, B. M. (2016) Remixing as a pathway to computational thinking.

DiSessa, A. A., & Cobb, P. (2004). Ontological innovation and the role of theory in design experiments. *The journal of the learning sciences, 13(1)*, 77-103. doi: 10.1207/s15327809jls1301_4

Duncan, C., Bell, T., & Tanimoto, S. (2014, November). Should your 8-year-old learn coding?. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, ACM, 60-69.

European Schoolnet (2014). Computing our future: Computer programming and coding- priorities, school curricula and initiatives across Europe. From: http://www.eun.org/publications/detail?publicationID=481

Field, A. (2009). *Discovering statistics using SPSS.* Sage publications. ISBN: 978-1-84787-906-6.

Flannery, L. P., Silverman, B., Kazakoff, E. R., Bers, M. U., Bontá, P., & Resnick, M. (2013). Designing ScratchJr: Support for early childhood learning through computer programming. In *Proceedings of the 12th International Conference on Interaction Design and Children*, ACM, 1-10.

Franklin, D., Hill, C., Dwyer, H., Iveland, A., Killian, A., & Harlow, D. (2015). Getting started in teaching and researching computer science in the elementary classroom. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, ACM, 552-557.

Jeuring, J., Corbalan, G., van Es, N., Leeuwestein, H., van montfort, J. (2016). Leren programmeren in het PO - een literatuurreview. NRO. From: https://www.nro.nl/kennisrotondevragenopeenrij/effecten-programmeeronderwijs-op-programmeervaardigheden/

Kafai, Y. B., & Resnick, M. (1996). Constructionism in practice: Designing, thinking, and learning in a digital world. New Jersey: Lawrence Erlbaum Associates.

Kalelioglu, F., & Gülbahar, Y. (2014). The effects of teaching programming via scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education, 13(1)*, 33.

Kirschner, P. A., Sweller, J., & Clark, R. E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational psychologist, 41(2)*, 75-86.

Lee, Y. J. (2011). Scratch: Multimedia programming environment for young gifted learners. *Gifted Child Today, 34(2)*, 26-31.

Lewis, C. M. (2010). How programming environment shapes perception, learning and goals: logo vs. scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education*, ACM, 346-350.

McHugh, M. L. (2012). Interrater reliability: the kappa statistic. *Biochemia Medica, 22(3)*, 276âĂŞ282.

Ngai, G., Chan, S. C., Leong, H. V., & Ng, V. T. (2013). Designing i* CATch: A multi-purpose, education-friendly construction kit for physical and wearable computing. *ACM Transactions on Computing Education (TOCE), 13(2).*

Papert, S., & Harel, I. (1991). Situating constructionism. *Constructionism*, Westport: Ablex Publishing, 1-11.

Poortman, S., & Sloep, P. (2006). Education models.

Sawyer, R. K. (2005). The Cambridge handbook of the learning sciences. Cambridge University Press.

Sun, S., Pan, W., & Wang, L. L. (2010). A comprehensive review of effect size reporting and interpreting practices in academic journals in education and psychology. *Journal of Educational Psychology, 102(4)*, 989 - 1004.

Van Merriënboer, J. J., Clark, R. E., & De Croock, M. B. (2002). Blueprints for complex learning: The 4C/ID-model. *Educational technology research and development, 50(2)*, 39-61, doi: 10.1007/BF02504993.

Van Merriënboer, J.J.G., & Dijkstra, S. (1997). The four-component instructional design model for training complex cognitive skills. *R.D. Tennyson, F. Schott, N.Seel & S. Dijkstra (Eds.),Instructional design: International perspectives. Theory, Research, and Models (Vol. 1).* Mahwah, New Jersey: Lawrence Erlbaum Associates, 427-445.

Van Merriënboer, J. J., & Kirschner, P. A. (2012). Ten steps to complex learning: A systematic approach to four-component instructional design. New York and London: Routledge.

Wilson, A., Hainey, T., & Connolly, T. (2012). Evaluation of computer games developed by primary school children to gauge understanding of programming concepts. In *European Conference on Games Based Learning*. Academic Conferences International Limited.