Universiteit Leiden

Psychologie
Faculteit der Sociale Wetenschappen

# Show and tell?
# The influence of instruction types on developing programming skills and self-efficacy levels in elementary school

Charlotte Thepass

Master Thesis School Psychology
Date: 02-07-2019
Student number: s1137247
Supervisor: Dr. Kiki Zanolie (& Dr. Felienne Hermans, LIACS)
Second reader: Michelle Achterberg

## Abstract

Research suggests the effectiveness of direct instruction over minimal instruction, even though many programming classes for elementary school use minimal instruction. We have examined the effect of instruction type on comprehension and skills, and self-efficacy in school classes, hypothesizing the effectiveness of direct instruction over minimal instruction. Two classes participated in the experiment (total N = 34, 17 girls, M age = 10.21; SD=0.60), attending six programming classes differing in instruction type, and making an exam. Self-efficacy was measured with the NPV-J-2. The group receiving minimal instruction performed significantly better on the exam, and reported significant higher levels of self-efficacy, suggesting the effectiveness of minimal instruction rather than direct instruction in skills and comprehension, and self-efficacy.

## Index

## 1. Introduction

In 2014, the Dutch parliament accepted a proposal that urged state secretary of education Sander Dekker to research what adjustments should be made for the curriculum of Dutch primary education. It was emphasized that skills on the curriculum should fit with the 21st century mindset. An important and new part of those skills are learning to program and the development of 'digital literacy'. Digital literacy is about awareness of reliability of source material, and the effects of the digital world on real life events (Dekker, 2014). Computer programming or programming is the process of writing a script or algorithm (tutorial) that can be read and executed by a machine, robot or tool (Jeuring, Corbalan, van Es & Leeuwestein, 2016). Now, five years later, a new curriculum for Dutch education will be presented to the parliament, including a section about programming education (Ontwikkelteam Digitale Geletterdheid, 2018). However, little substantial research has been conducted on computer programming education in primary schools (Waite, 2017). With this thesis, we want to add to existing literature about programming education. Our research focuses on instructional methods in programming education and its effect on comprehension of basic computer programming concepts, and computer programming skills in Scratch. We will compare a type of direct instruction, more specifically explicit direct instruction, with a type of minimal instruction, more specifically the Exploratory Learning Model.

Through the years, several direct instruction models have been developed (e.g., Bereiter & Engelmann, 1966; Good & Grouws, 1979; Hunter, 1982). However, there are six overlapping components which nearly all DI-models share: (1) material must be broken down into small steps and put in a logical order, (2) objectives are learner or performance focused, (3) reactivating what students already know and connecting it with their new knowledge is emphasized, (4) students practice every step or combination of steps, (5) the promotion of additional practice opportunities, and (6) feedback moments throughout the lesson (Magliaro, Lockee & Burton, 2005). These components can also be found in explicit direct instruction, a group of research-supported instructional behaviours that are used to support successful learning through clarity of language and purpose, and reduction of cognitive load. Active student engagement is promoted by requiring response followed by affirmative and corrective feedback (Hughes, Morris, Therrien & Benson, 2017). Explicit instruction has similarities with Engelmann's (Bereiter & Engelmann, 1966) well known Direct Instruction model, and is based on five pillars: (1) segment complex skills, to make them more 'manageable', (2) providing clear, and consistent descriptions of how the skill or strategy is formed, (3) promoting successful engagement by giving fading support/prompts, (4) providing opportunities for students to respond and receive feedback, and (5) creating purposeful practice opportunities. Research has shown the effectiveness of direct instruction, especially when teaching novel learners (van Merriënboer & Kirschner, 2007; Lister, 2016). Since programming education in elementary schools is a relatively new subject, it can be assumed that children have little to no prior experience. When a novel learner has to discover the concept by themselves, misconceptions can easily be made, which may interfere with later learning (Kikas, 2004; Clark, Kirschner & Sweller, 2012). Additionally, direct instruction benefits weak learners, because it reduces cognitive load (Brown & Campione, 1994; Smith, Sáez & Doabler, 2016; Hughes et al., 2017). According to the cognitive load theory, instructional information is processed in the working memory (Sweller, Ayres & Kalyuga, 2011). Since working memory is limited in the information it can process in a certain amount of time (Cowan, 2001) and exploration of complex and novel environments may generate heavy working memory load, thus negatively affecting learning in students (Kirschner, Sweller & Clark, 2006). Furthermore, in literary reviews, direct instruction is found to be more effective than minimal instruction (Mayer, 2004; Hattie, 2009, Jeuring et al., 2016). Therefore, teaching computer programming with direct instruction is a research-supported approach and we believe that it should be the default when teaching programming classes.

Instead, the opposite is true: in computer programming education, constructivism is a popular way of teaching. According to this theory, students must construct their own mental representations of the world through active cognitive processing (Clark, et al., 2012), this way of teaching uses explorative learning minimal explanations or guidance from the teacher. Constructivism in computer programming education was popularized in the 80’s by Seymour Papert. As Lister said in his paper ‘Toward a Developmental Epistemology of Computer Programming’ (2016, p. 6): ‘We are all constructivists now.’ In this study, we will use the definition for exploratory learning as a counterpart for direct instruction. Exploratory learning is defined as learning through exploring real or virtual environments combined with peer or tutorial support (de Freitas, 2006) and is based on the idea of constructivism. The exploratory learning model (ELM; de Freitas & Neumann, 2009) is based on Kolb’s learning cycle for explorative learning (1984). An important step of Kolb’s learning cycle is the concrete (real-life) experiences. However, in the current educational landscape, digital learning contexts cannot be ignored. The ELM acknowledges abstract, real life (lived), and virtual experiences as definitive experiences (de Freitas & Neumann, 2009). In the ELM, reflection (3) is added after exploration (see Fig 1). In the current model, establishing learning transfer between the different types of experience is very important. Adding a step of Reflection, allows the student to consider what they have learned and supports the forming of abstract concepts and testing in different situations (de Freitas & Neumann, 2009).

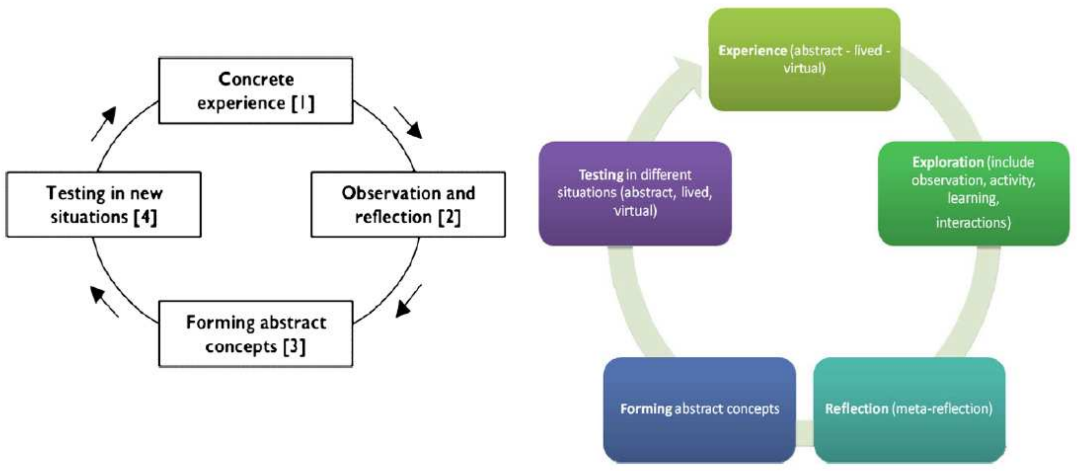

Concrete experience [1]
Observation and reflection [2]
Forming abstract concepts [3]
Testing in new situations [4]
Experience (abstract - lived - virtual)
Exploration (include observation, activity, learning, interactions)
Reflection (meta-reflection)
Forming abstract concepts
Testing in different situations (abstract, lived, virtual)

**Fig 1.** Kolb's experiental learning cycle (Kolb, 1984) versus the Exploratory Learning Model (de Freitas & Neumann, 2009)

A benefit of explorative learning is the positive influence on motivation, which plays an important part in acquiring programming skills (Bergin & Reilly, 2017). Students tend to choose explorative learning over direct instruction, even though it may increase the difficulty of the assignment (Clark, et al., 2012). Also, explorative learning mimics the tasks of a programmer, which often requires thinking outside of the box, and explorative and testing (Boyer, Langvin & Gaspar, 2008). However, explorative learning does increase the development of misconceptions, which has a negative effect on academic success (Ben-Ari, 2001). Even though explorative learning may have small, positive effects on developing skills, direct instruction has larger effects (Klahr & Nigam, 2004), and, it is important for students to receive guidance and feedback from teachers (Mayer, 2004; Tobias & Duffy, 2009; Alfieri, Brooks, Aldrich & Tenenbaum, 2011; Hsiao, Lin & Kang, 2011). Therefore, we believe that pure exploratory learning is an ineffective method to teach computer programming.

While many studies explored the effects of instruction type on academic performance in computer programming (Ramalingam, Labelle, & Wiedenbeck, 2004; Boyer, et al., 2008; Ismail, Ngah & Umar, 2010), research on the effects of instruction type in children (under 18) is lacking. Waite (2017) showed in a review that there is limited empirical evidence to support advice on programming education in schools. Also, the research that has been conducted with primary and high school students, was often small scaled, in out of school settings, and had short time frames. Thus, so far, most studies have found in adults a positive effect of direct instruction on programming skills, however, only limited empirical evidence is found for children (under 18). Also, very few studies have been  conducted in the Netherlands, even though there are many Dutch programming courses taught in primary and high schools.

Dutch programming courses for children have been developed in the last three decades. However, most courses have been made in the last five years (Strijker, 2018). There are roughly four different categories within these programming courses: visual, textual,

Table 1.
*Programming categories, definitions (Jeuring, et al., 2016)*

| Category | Description | Programming language |
|---|---|---|
| Textual | Code is written text, underlying grammar is important. | e.g. Java, C, Python |
| Visual | Code is built by moving blocks on screen, blocks are often represented as puzzle pieces. | e.g. Scratch, code.org |
| Unplugged | Code is warren without any electronics (‘unplugged’) | e.g. csunplugged.org |
| Physical | Code is made using physical blocks, this makes it easier for students to visualize different steps and variables in a script. Often used in kindergarten. | e.g. Lego Mindstorms, Dash & Dot |

‘unplugged’, and physical programming (see Table 1.).

However, the programs that are offered often use explorative learning as instructional method (see Table 2.), even though research has shown that this type of instruction is an ineffective way to teach programming (Mayer, 2004; Kirschner, et al., 2006; Clark, et al., 2012). As such, it is of great importance to investigate the effects of instruction type on programming skills in a younger sample.

Table 2.

*Distribution of programming courses for elementary school by programming category (Thepass, C., 2019)*

| Category | N | Most common grade (n) | Most common instructional method (n) |
|---|---|---|---|
| Textual | 13 | 6, 7, 8 (12) | Explorative (9) |
| Visual | 17 | 7 (12) | Explorative (16) |
| Unplugged | 6 | 6, 7 (6) | Explorative (5) |
| Physical | 3 | 2 (3) | Explorative (3) |
| Mix | 10 | 6, 7, 8 (9) | Explorative (7) |

Another important factor that influences the development of computer programming skills is the self-efficacy of students. Self-efficacy, first described by Bandura (1993), is the belief that one has the abilities to achieve one's goals and complete tasks. This belief in abilities originates from verbal persuasion from an authority, previous successes within a certain domain, observing success in a peer with (perceived) similar capabilities, and the physiological response to the task (Hushman & Marley, 2015). Several factors influence self-efficacy with regard to computer skills, such as: parental support (Vekiri & Chronaki, 2008), previous experience (Aivaloglou & Hermans, 2019), and following programming classes (Ramalingam, et al., 2004). Self-efficacy has a positive effect on academic achievement (Britner & Pajares, 2006), because of its self-regulating function. Students with high self-efficacy levels are better at regulating their impulses when they fail at a task, because they have the belief that they are able to finish the task, even if there is a setback during the task (Komarraju & Nadler, 2013). Again, most research regarding self-efficacy in computer programming skills focused on adults using a self-efficacy questionnaire (Cassidy & Euchus, 2002). However, it is not yet understood what the effect of instructional method might be on computer self-efficacy in children. Even though the implications and evidence are used for children's education. As such the current study will focus on the effect of instruction type on the self-efficacy with regards to computer programming skills in children.

This study researches instructional methods for computer programming courses in schools for children aged 8-11. There are two research questions guiding this study. First, do children who receive direct instruction in a programming class acquire better skills and have a better comprehension of computer programming than children who receive minimal instruction? Research suggests the effectiveness of direct instruction over minimal instruction (Mayer, 2004; Kirschner, et al., 2006, Hattie, 2009; Clark, et al., 2012). Therefore, it is hypothesized that children who receive direct instruction in computer programming class, acquire better computer programming skills and have a better comprehension of computer programming than children who receive minimal instruction. Children with no prior knowledge of the subject perform better if they receive direct instruction, because the guidance directs their attention to the important aspects of programming and enables them to think in the most efficient way (Jeuring, et al., 2016; Lister, 2016).

The second question guiding this thesis is: Do children who receive direct instruction during a programming class have a higher self-efficacy after the course is over than children who receive minimal instruction? Research suggests the positive influence of guided instruction on science self-efficacy and receiving persuasion from an authority is an important factor for self-efficacy (Hushman & Marley, 2015). Therefore, it is hypothesized that children who receive direct instruction in a programming course, have a higher self-efficacy at the end of the course than children receiving minimal instruction.

## 2. Method

### *2.1 Participants*

Participants were 35 students (17 girls, M age = 10,21; SD=0,60) currently in 5th grade (groep 6) or 6th grade (groep 7), from two different schools. One group of students (5th grade, n=15, seven girls, M age = 9180; SD=0,41) attended an elementary school in the Hague. The other group of students (6th grade, n=20, 11 girls, M age = 10,50; SD=0,51) attended an elementary school in Rijnsburg, near Leiden. Both schools had participated in computer programming education research before and were contacted again for this research. 33% of the 5th graders, and 80% of the 6th graders had prior programming experience. Three students from the Hague had used Scratch before, none of the students from Rijnsburg had used Scratch before. The group of students from Rijnsburg had participated in a Lego Mindstorms project, the week before the experiment started. However, since that was a singular day project, some students did feel they had no prior experience. Therefore, we assume that participants had little to no prior computer programming experience. Participants were divided over two experimental conditions, based on which school they attended. Schools were randomly assigned a type of instruction by entering both names into an online list randomizing generator. Five children (four boys, one girl) from the DI group did not take the final test, because they left the experiment (DI group, N=10). The subjects of this study were under the age of 16, therefore one or more caregivers gave informed consent. The research was approved by the local ethics committee at Leiden University. Participants did not receive compensation for this study.

### *2.2 Measures*

#### *Demographics and prior knowledge*

Students filled in a demographic questionnaire, including questions about their age and gender. To measure pre-existing knowledge of computer programming, students answered questions about their experience with computer programming. They were also asked to describe what a programmer does. The amount of correct answers were added together to produce a pre-test score, including an extra point for prior experience. In total, four points could be obtained.

#### *Comprehension and skills*

To measure the comprehension and knowledge of the computer programming concepts, students made a test which was created by the researcher of this study, because no standardized computer skills questionnaire for children exists. The test is based on materials by Hermans & Swidan  (2019). No Crohnbach's Alpha is available, but experts in the educational and computer programming field have reviewed the test questions. Students answered ten multiple choice questions, with four answer options each. The test measured five different concepts: algorithms, variables, for-loops, conditional statements, and debugging. For every concept, one question was about the definition of the concept, the other question was about applying the concept in Scratch (see Appendix A for the distribution of the questions). The amount of correct answers were added together to produce a total test score, with a range from zero to ten points. Also, participants could score 2 points per concept, and a skill score and comprehension score could be calculated, with a total score between zero to five points each.

#### *Self-efficacy*

##### *Pre-test*

To measure the base level of self-efficacy, three (sub)scales of the Dutch version of the Motivational Strategies for Learning Questionnaire (MSLQ) were used: task value (Crohnbach's Alpha: .,90), self-efficacy (Crohnbach's Alpha: ,93) , and help seeking (Crohnbach's Alpha: ,52). One of the items is 'I believe that I will achieve high grades for this programming course'. The MSLQ is a self-report instrument that measures motivational orientations and use of different learning strategies for a course (Pintrich, Smith, Garcia & McKeachie, 1991). The Dutch version has been published by Severiens (1999). This instrument has 44 items divided over five scales: Intrinsic value, Self-efficacy, Test anxiety, Strategy use, and Self-regulation on a 7-point Likert-scale. For every scale, the mean of the items that make up the scale is calculated. Some items are reversed. No norm scores exist, the questionnaire is in reference to the specific group that is taking the questionnaire.

*Post-test*

To measure self-efficacy, two subscales of the Nederlandse PersoonlijkheidsVragenlijst-Junior-2 (NPV-J-2) were used: Inadequacy (Crohnbach's Alfa: .92;), and Tenacity (Crohnbach's Alpha: ,84). The scale Inadequacy measures if a child thinks negatively about themselves. One of the items is: 'I feel fine most of the time'.  The scale Tenacity measures if a child thinks it's tenacious. One of the items is: 'I prefer to work orderly'. The NPV-J-2 is the revised, adapted version of the PersoonlijkheidsVragenlijst (2011). The NPVJ-2 has 100 items, divided equally over five scales: Inadequacy, Tenacity, Social inadequacy, Stubbornness, and Dominance. Participants self-report to what extend they identify with the statement. They can chose from three different answers: Yes (2 points), ? (1 point) and No (0 points). For every scale, all relevant item scores were added together to produce a raw score. This score can be interpreted as being extremely low to extremely high. The total raw score per scale is between 20 and 40 points (Barelds, Luteijn & van Dijk, 2011).

*Scratch*

Students were taught programming skills in the online program Scratch. Scratch is a block-based programming language, developed by the Lifelong Kindergarten Group from MIT Media Lab and specifically designed for children from age 8-16 (Resnick, et al., 2009). In Scratch, sprites which are displayed on a stage can be controlled via scripts. Scripts are created by dragging and dropping blocks in the assigned programming space. Blocks are colour coded and represent different program components, such as variables and conditions. Blocks need to be connected like puzzle pieces to create a script, and a sprite can follow several scripts at once. (Meerbaun-Salant, Armoni & Ben-Ari, 2013).

### *2.3 Design*

This study examines the relation between ‘instructional method’ and improving the knowledge and comprehension of computer programming, and ‘method of instruction’ and self-efficacy. To do this, direct instruction and exploratory learning were compared within the context of learning how to program. A quasi-experimental method was used for this research.

### *2.4 Procedure*

Students attended six programming classes, that were about an hour long. In the first five weeks, they learned one new programming concept per week. The pre-test, a demographic questionnaire and the Dutch version of the MSLQ, was filled in by students during the first class. The post-test, the test that we created and based on a test by Swidan & Hermans (2019), and the NPV-J-2, were filled in during the sixth and final programming class. The classes were experimenter created, based on online materials (Hermans, F., n.d.). The students learned programming via Scratch. Both groups differed in instruction method, the first group received Direct Instruction (DI group) and the second group received minimal instruction (MI group). Each lesson had four parts: introduction of the topic, unplugged assignment, assignment in Scratch, ending. The DI group received instruction based on the Direct Instruction model: during the introduction, knowledge was reactivated and the topic of the week was explained. Students were taught which strategies they should use during the assignments. The unplugged and Scratch assignment were led by guided practice. Students were encouraged to try the program and feedback was given after every part of the class. The MI group followed the exploratory model, they had to work in groups during the unplugged assignment and were only given the blocks, but not the useful strategies for the Scratch assignment. The teacher of the MI group asked questions, to enhance (meta-)reflection in students, but did not give feedback on the assignments until the end of the class. The direct instruction classes were given by the researcher, the minimal instruction classes were led by the ICT employee of their school. Instructional booklets and additional information was given to the ICT employee beforehand. Both groups made the Scratch assignments in the assigned computer rooms of their school. The DI group worked on desktops and Chromebooks. Most students preferred the desktop, since the Chromebooks had smaller screens and a slower internet connection. Only seven desktops were available, therefore, about half of the students had to work on the Chromebooks. In the MI group, the unplugged assignment took place in their own classroom. The computer room had a working desktop computer for each student.

### *2.5 Statistical Analysis*

Categorical variables were reported in number and percentages, and analyzed with chi-square tests. Continuous variables were reported in terms of mean and standard deviations, and analyzed with independent t-test or Mann-Whitney tests, when statistical assumptions were violated. Given the way the hypotheses were set up, a significance level of p <.025 (1-tailed), was established. Since a high score on the scale Inadequacy indicates a low self-esteem, this variable was reverse coded, to make sure a high score on the scale indicated a high self-esteem. The effect size and power were calculated, using an alpha level of 0,05. A Pearson R was used to calculate the relationship between the MSLQ and the NPV-J-2.

## 3. Results

### *3.1 Demographics and internal consistencies*

To test whether there were more boys or girls in the Direct Instruction (DI group) compared to the Minimal Instruction (MI) group, we performed a CHI-squared test (see Table 3.), which indicates no between-group differences in gender, $\chi^2 = .038$, *p = .794*. To test whether there was an age difference between the DI group (µ = 9.8) and the MI group (µ = 10.5), we performed an independent t-test, which indicates a between-group difference in age, t = -3.980, *p = .001*. To test whether there was a difference in self-efficacy levels on the MSLQ, we performed an independent t-test, which indicates no between-group difference in self-efficacy, t = -1.881, *p = .071*. However, the variable ‘Prior experience’, t = -4.050, p = .000 showed a between-group difference in prior experience, such that the MI group had more experience than the DI group. Since the groups differed significantly in age and prior experience, the hypotheses were tested with nonparametric tests. The Pearson correlation between the NPVJ-scales, and MSLQ scales was .257, this is a weak positive relationship, indicating that a high score on the MSLQ-scales equals a high score on the NPVJ scales. Effect size was calculated with Hedges’ g = 1.52639 for total test score, indicating an invalid effect size. The effect size for the total MSLQ score was g = 0.989625, indicating a large effect size.

Table 3.

*Characteristics participants*

| | | DI group N = 10 | | MI group N=20 | | $\chi^2$ (df=1) | p |
|---|---|---|---|---|---|---|---|
| | | Frequency | Percent | Frequency | Percent | | |
| Gender | | | | | | .068 | .794 |
| | Boy | 4 | 40 | 9 | 45 | | |
| | Girl | 6 | 60 | 11 | 55 | | |
| | | | | | | T | p |
| | | µ | SD | µ | SD | | |
| Age | | 9.80 | .422 | 10.50 | .513 | -3.980 | .001 |
| Prior experience | | .44 | .726 | 1.89 | 1.15 | -4.050 | .000 |
| Missing | | 1 | | 1 | | | |
| MSLQ | | 4.8086 | 1.22 | 5.50 | .733 | -1.881 | .071 |

### *3.2 Knowledge and skills*

To test the hypothesis that children who received direct instruction in a programming class would acquire better programming skills and a better comprehension of computer programming compared to children who received minimal instruction in a programming class, we performed a series of Mann-Whitney U tests. Contrary to our expectations, we found that the MI group (µ = 6.4) achieved a higher score than the DI group (µ = 3.7), *U* = 26.500, *p =.001.* Furthermore, the MI group performed significantly better on the variables skills (MI group: µ = 2.8, DI group: µ = 1.4, *U* = 41.500, p = .003) and comprehension (MI group: µ = 3.625, DI group: µ = 2.3, *U* = 44.000, p = .006), disproving our hypothesis even more.

### *3.3 Self-efficacy*

To test the hypothesis that children receiving direct instruction in programming classes would have higher scores on the NPV-J-2 scales than children receiving minimal instruction, we performed a series of Mann-Whitney U tests. Contrary to our expectations, we found that the MI group (µ = 3.35) reported higher levels of self-efficacy than the DI group (µ = 1.9), *U* = 42.000, *p = .004.* Furthermore, there was no difference on the scale Tenacity between the DI group (µ = 3.20) and the MI group (µ = 3.60), *U* = 83.500, *p = .23*).

## 4. Discussion

The present study predicted that explicit direct instruction in elementary school computer programming classes would be a more effective instructional method than the exploratory learning model, comparing the variables computer programming comprehension, computer programming skills, and self-efficacy. However, no evidence was found to support these hypotheses. Although, the type of instruction did seem to affect the comprehension and skills, and self-efficacy levels, after a 6-week training in computer programming skills, such that students receiving minimal instruction showed a higher level of comprehension and skills, and self-efficacy. These findings imply that not direct instruction, but minimal instruction is an improving factor of computer programming comprehension and skills, and computer self-efficacy.

These findings are surprising, because earlier research suggests that novel programmers should receive direct instruction (Lister, 2016), and the effectiveness of direct instruction over minimal instruction (Mayer, 2004; Kirschner, et al., 2006). However, we did not use a pure exploratory way of teaching the MI group, the teacher provided feedback and asked questions to help the student understand the programs, which has been found to be an effective method of teaching (Tobias & Duffy, 2009; Hsiao, et al., 2011). Yet, this was not reflected in the results of the DI group, where students also received a lot of feedback. An explanation for this difference can be found in the used materials: the MI group had access to working PC's with large screens, while half of the DI group used chromebooks with small screens and slow working internet. This did not only lead to frustration among the students, but also made it more difficult to finish the assignments. Furthermore, this increased the cognitive load of the students who had to regulate their emotions and learn in a new environment. According to Kirschner, et al. (2006) this has a negative effect on learning in students.
Our findings on self-efficacy also show a significant difference between both groups, in favour of the MI group. One explanation for this finding is the difference in prior experience between both groups. The MI group had participated in a Lego Mindstorms lesson, the week before the experiment. Even though this is different from Scratch, the students had already practiced with working in a computer programming environment and experienced the different mindset that is needed for programming. Also, when answering the question: 'What does a programmer do?', the MI group were able to fill in more correct answers than the DI group.

Even though following a programming class has an impact on self-efficacy (Ramalingam, et al., 2004), succeeding in a task and observing success in classmates is also important (Hushman & Marley, 2015). The badly working materials in the DI group resulted in poorly executed assignments, thus, students did not succeed in the tasks and did not see their classmates succeed, which may have decreased their self-efficacy levels. Moreover, both groups reported similar levels of tenacity, thus, both groups felt like they worked really hard. The MI group saw that their hard work paid off, the DI group did not, which may have also decreased self-efficacy levels. Furthermore, using materials that do not work, also affect self-efficacy (Hsu & Huang, 2006).
Perhaps, both comprehension and knowledge (focus of the first hypothesis) and self-efficacy (focus of the second hypothesis) influenced each other. As noted above, succeeding in tasks influences self-efficacy (Hushman & Marley, 2015), but self-efficacy has also been found to influence student performances (Parajes & Graham, 1999). Our results show a similar effect: the group that performed better (MI group), also had higher self-efficacy levels.

*Limitations*

However, we cannot say that these differences in results relied strictly on instructional types or other explanations. Other factors may also have resulted in different findings. Since one of the classes was taught by the researcher, who has limited teaching experience, the lessons evolved along the way. In week 3-5 the Scratch assignment was to build games. Even though the children did enjoy these classes over the earlier classes, it also meant less focus on the concept of the week. Additionally, this study used a relatively small sample size (total N = 35), which reduced the power of the research. Thus, lowering the probability of finding true effects, and positive predictive value, and increasing the effect of errors (Button, et al., 2013). Furthermore, the lack of standardized testing and teaching materials results in researcher created materials, which lessens the overall validity and especially the construct validity of the research. It is however a common phenomenon in computer education research to create and use researcher designed materials in own studies (e.g. Feaster, Ali, Zhai & Hallstrom, 2014; Benotti, Martínez & Schapachnik, 2014), maybe, because it is a relatively new research field and standardized protocols do not yet exist (as they do in psychology research).

*Future recommendations*

To our knowledge, this research is among the first studies examining the effect of instruction type on self-efficacy in children in Dutch elementary school computer programming classes. Thus, further work is required in order to gain a more complete understanding of the influence of instructional methods on programming skills, especially with the implementation of computer programming in the official Dutch curriculum. A topic that is suitable for further research is examining the effects of motivation on learning ability in computer programming context. We noticed in our groups a difference in motivation: a female student from the MI group talked about how much she loved the programming class, whilst in the DI group, children talked about quitting the experiment after the second class. Since intrinsic motivation plays an important part in learning to program (Bergin & Reilly, 2017), this may have influenced both groups in their performance. As noted before, many researchers create their own materials, which decreases the validity of the research. Therefore, the next step in research should be creating standardized test materials and protocols for computer educational research. In line with Waite (2017), this field of research needs more longitudinal research with large sample sizes on the pedagogy behind programming education.

In summary, the evidence from this study indicates that minimal instruction is more effective than direct instruction in computer programming classes for elementary schools. However, taking the small sample size of the study into account, the current results should be interpreted with caution. Additional experimental research is needed to examine the effectiveness of instruction types on computer programming skills and self-efficacy in children. If a child needs to learn how to read, it would not be handed a book and told to learn reading by himself. It should be the same for programming education.

## References

Aivaloglou, E. & Hermans, F. (2019). Early Programming Education and Career Orientation: the Effects of Gender, Self-efficacy, Motivation and Stereotypes. Proceedings of the 50th ACM Technical Symposium on Computer Science Education, New York, United States.

Alfieri, L., Brooks, P., Aldrich, N. & Tenenbaum, H. (2011). Does Discovery-Based Instruction Enhance Learning? *Journal of Educational Psychology, 103*(1), 1-18.

Bandura, A. (1993) Perceived Self-Efficacy in Cognitive Development and Functioning. *Educational Psychologists, 28*(2), 117-148.

Barelds, D., Luteijn, F. & van Dijk, H. (2011) *NPVJ-2. Junior Nederlandse Persoonlijkheids Vragenlijst.* Amsterdam: Pearson Assessment.

Ben-Ari, M. (2001). Constructivism in Computer Science Education. *Journal of Computers in Mathematics and Science Teaching, 20(*1), 45-73.

Benotti, L., Martínez M. & Schapachnik, F. (2014). Engaging high school students using chatbots. Proceedings of the 2014 conference on Innovation & technology in Computer Science Education, Uppsala, Sweden.

Bereiter, C. & Engelmann, S. (1966). *Teaching disadvantaged children in the preschool.* Upper Saddle River, NJ: Prentice Hall.

Bergin, S. & Reilly, R. (2017). The Influence of motivation and comfort-level on learning to program. In: P. Romera, J. Good, E. Acosta Chaparro & S. Bryant (Eds). Proc. PPIG 17.

Boyer, N. Langevin, S. & Gaspar, A. (2008). Self-Direction & Constructivism in Programming Education. *SIGITE 2008,* October 16-18, Cincinnati, OH, USA.

Britner S. & Pajares F. ( 2006) Sources of science self-efficacy beliefs of middle school students. *Journal of Research in Science Teaching, 43*(5), 485-499.

Brown, A. & Campione, J. (1994) Guided Discovery in a Community of Learners. In: K. McGilly (Ed.), *Classroom Lessons: Integrating Cognitive Theory and Classroom Practice* (pp. 229-270), Cambridge, MA: MIT Press.

Button, K. Ioannidis, J., Mokrysz, C., Nosek, B., Flint, J., Robinson, E. & Munafò, M. (2013) Power failure: why small sample size undermines the reliability of neuroscience. *Nature Reviews Neuroscience, 14,* 365-376.

Cassidy, S. & Euchus, P. (2002). Developing the Computer User Self-efficacy (Cuse) Scale: Investigating the Relationship between Computer Self-efficacy, Gender and Experience with Computers. *Journal of Educational Computing Research, 26*(2), 133-153.

Clark, R., Kirschner, P. & Sweller, J. (2012) Putting students on the Path to Learning: The Case for Fully Guided Instruction. *American Educator*, *36*(1), 6-11.

Cowan, N. (2001). The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, *24*, 87–114.

Dekker, S. (2014-11-27). Toekomstgericht funderend onderwijs. *Tweede Kamer der Staten-Generaal.*

Feaster, Y., Ali, F., Zhai, J. & Hallstrom, J. (2014) Serious Toys: three years of teaching computer science concepts in K-12 classrooms. Proceedings of the 2014 conference on Innovation & technology in Computer Science Education, Uppsala, Sweden.

Freitas, S. de (2006). Using games and simulations for supporting learning. In C. Martin & L. Murray (Eds.), *Learning, media and technology special issue on gaming*. 31(4), 343-358.

Freitas, S. de & Neumann, T. (2009) The use of 'exploratory learning' for supporting immersive learning in virtual environments. *Computers & Education, 52*, 343-352.

Good, T. L. & Grouws, D. A. (1979). The Missouri mathematics effectiveness project. *Journal of Educational Psychology*, *71*(3), 355-362.

Hattie, J. (2009). *Visible learning: A synthesis of 800+ meta-analyses on achievement*. New York: Routledge.

Hermans, F. (n.d.) Scratchlessen. Via: www.scratchles.nl

Hermans, F. & Swidan, A. (2019) The Effect of Reading Code Aloud on Comprehension: An Empirical Study. *ACM Global Computing Education Conference CompEd 2019*, Chengdu, China.

Hsiao, S., Lin, J. & Kang, J. (2011) Learning to program in kpl through guided collaboration. *US-China Education Review*, *8*(1), 89-97.

Hsu, W. & Huang, S. (2006). Determinants of computer self-efficacy – an examination of learning motivations and learning environments. *J. Educational Computing Research*, *35*(3), 245-265.

Hughes, A., Morris, J., Therrien, W. & Benson, S. (2017). Explicit Instruction: Historical and Contemporary Contexts. *Learning Disabilities Research & Practice, 32*(3), 140-148.

Hunter, M. (1982). *Mastery teaching.* El Segundo, CA: Theory Into Practice.

Hushman, C. & Marley, S. (2015). Guided Instruction Improves Elementary Student Learning and Self-Efficacy in Science. *The Journal of Educational Research*, *108*(5), 371-381.

Ismail, M., Ngah, N. & Umar, I. (2010). Instructional Strategy in the Teaching of Computer Programming: A need Assessment Analyses. *TOJET: The Turkish Online Journal of Educational Technology*, *2010*, *9*(2), 125-131.

Jeuring, J., Corbalan, G., van Es, N. & Leeuwenstein, H. (2016). Leren programmeren in het PO – een literatuurreview.

Via https://www.nro.nl/kennisrotondevragenopeenrij/effecten-programmeeronderwijs-op-programmeervaardigheden/

Kikas, E. (2004) Teachers' Conceptions and Misconceptions Concerning Three Natural Phenomena, *Journal of Research in Science Teaching 41*(5) 432–448.

Kirschner, P., Sweller J. & Clark, R. (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, 42*(2), 75-86.

Klahr, D., & Nigam, M. (2004). The equivalence of learning paths in early science instruction: effects of direct instruction and discovery learning. *Psychological Science*, *15*, 661–667.

Kolb, D. A. (1984). Experiential learning. Englewood Cliffs, NJ: Pearson Ft Press.

Komarraju, M. & Nadler, D. (2013). Self-efficacy and academic achievement: Why do implicit beliefs, goals, and effort regulation matter? *Learning and Individual Differences*, *25*, 67-62.

Lister, R. (2016). Toward a Developmental Epistemology of Computer Programming. Proceedings of the 11th Workshop in Primary and Secondary Computing  Education, 5-16, Germany, Münster.

Magliaro, S., Lockee, B. & Burton, J. (2005). Direct instruction revisited: A key model  for instructional technology. *Educational Technology Research and Development, 2005, 53*(4), 41-55.

Mayer, R. (2004) Should There be a Three-Strikes Rule Against Pure Discovery Learning? *American Psychologist*, *59*(1), 14-19.

Meerbaun-Salant, O., Armoni, M. & Ben-Ari, M. (2013). Learning computer science concepts with Scratch. *Computer Science Education*, 23(3), 239-264.

Merriënboer, J. van, & Kirschner, P. (2007) Ten Steps to Complex Learning. Mahwah, NJ: Lawrence Erlbaum Associates.

Digitale Geletterdheid (2018) Ontwikkelteam Digitale Geletterdheid. Via https://curriculum.nu/ontwikkelteam/digitale-geletterdheid

Parajes, F. & Graham, L. (1999). Self-Efficacy, Motivation Constructs, and Mathematics Performance of Entering Middle School Students. *Contemporary Educational Psychology*, *24*, 124-139.

Pintrich, P., Smith, D., Garcia, T. & McKeachie, W. (1991) A manual for the Use of the Motivated Strategies for Learning Questionnaire (MSLQ). Ann Arbor, Michigan: University of Michigan.

Ramalingam, V., Labelle, D. & Wiedenbeck, S. (2004). Self-efficacy and mental models in learning to program. *ACM SIGCSE Bulletin, 36*(3), 171-175.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. & Kafai, Y. (2009) Scratch: Programming for All. *Communications of the ACM, 52*(11), 60-67.

Severiens, S. (1999). *Vragenlijst Motivatie, Leerproces en Leerstrategieën.* Amsterdam: Universiteit van Amsterdam.

Smith, J. L. M., Sáez, L., & Doabler, C. T. (2016). Using explicit and systematic instruction to support working memory. *TEACHING Exceptional Children*, *48*(6), 275–281.

Strijker, A. (2018). Voorbeeldmaterialen computational thinking [blog]. Via: http://curriculumvandetoekomst.slo.nl/21e-eeuwse-vaardigheden/digitale-geletterdheid/computational- thinking/voorbeeldmaterialen

Sweller, J., Ayres, P. & Kalyuga, S. (2011). *Cognitive Load Theory.* New York, NY: Springer.

Tobias, S., & Duffy, T. M. (Eds.). (2009). Constructivist theory applied to instruction: Success or failure? New York, NY: Taylor & Francis.

Vekiri, I. & Chronaki, A. (2008) Gender issues in Technology use: Perceived social support, computer self-efficacy and value beliefs, and computer use beyond school. *Computers & Education, 51*,1392-1404.

Waite, J. (2017) Pedagogy in teaching computer science in schools: a literature review. Queen Mary University of London, London.

## Appendices

### Appendix A

Distribution of concepts in the final test

Q 1. Algorithm – comprehension

Q 2. If-statements – comprehension

Q 3. For-loop – comprehension

Q 4. Variables – comprehension

Q 5. Algorithm – skill

Q 6. If-statement – skill

Q 7. Debugging – comprehension

Q 8. Variable – skill

Q 9. Debugging

Q 10. For-loop - skill

### Appendix B

Test questions

**1. Wat is een algoritme?**

A. Een regel waar de computer zich aan moet houden.

B. Een stappenplan voor de computer.

C. Een doosje waar de computer informatie in bewaart.

D. Een stuk code dat herhaald wordt.

**2. Tijdens de lessen heb je *voorwaardes* in Scratch gezet.**

**Zet een cirkel om de zin of zinnen die waar zijn.**

1. Een voorwaarde schrijf je met de woorden ‘als’ en ‘dan’.

2. Een voorwaarde wordt **altijd** herhaald.

**3. Wat is een herhaling bij programmeren?**

A. Een stappenplan voor de computer.

B. Een stuk code dat meerdere keren gedaan wordt.

C. Het oplossen van fouten in een code.

D. Tien keer dezelfde blokjes in het algoritme zetten

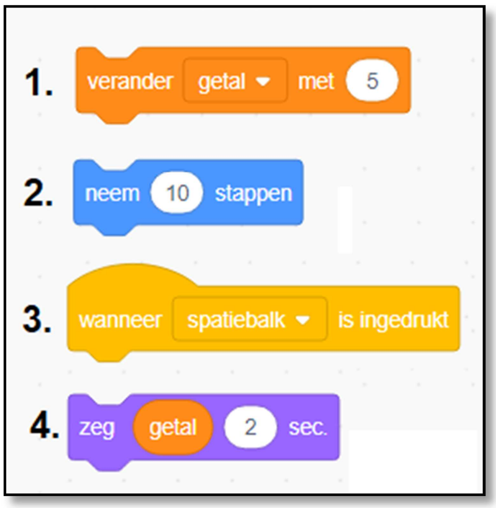
**4. Tijdens de lessen heb je *variabelen* gebruikt in Scratch.**

**Zet een cirkel om de zin of zinnen die waar zijn.**

1. Variabele gebruik je om een kortere code te kunnen schrijven.

2. Een variabele is een doosje waar de computer informatie in bewaard.

**5. Elke keer dat je op de spatiebalk drukt moet het poppetje lopen en een getal in de tafel van 5 zeggen. Wat is de juiste volgorde?**



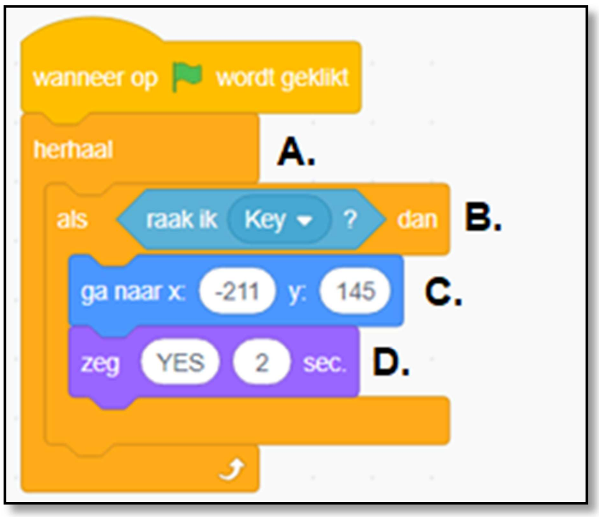
A. 1-2-3-4

B. 3-4-1-2

C. 1-4-2-3

D. 3-2-4-1

**6. Wat is de voorwaarde in dit stappenplan? (zet een cirkel om de juiste letter)**



**7. Wat is *debuggen*?**

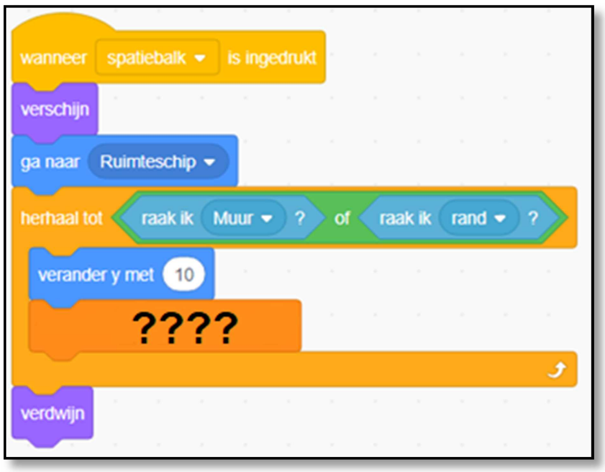
A. Tien keer dezelfde blokjes in het algoritme zetten.

B. Een stuk code dat herhaald wordt.

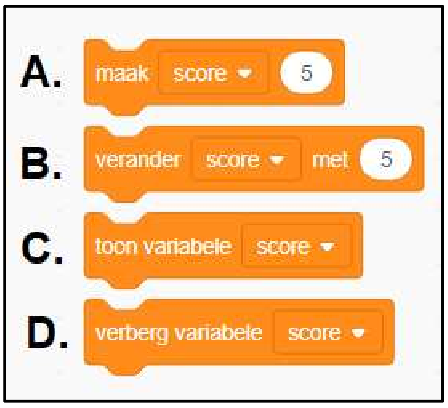
C. Het oplossen van fouten in een code.

D. Een kever uit de computer halen.

**8. Je wilt dat de score met 5 omhoog gaat. Welk blokje moet je gebruiken op de plek van het vraagteken??**

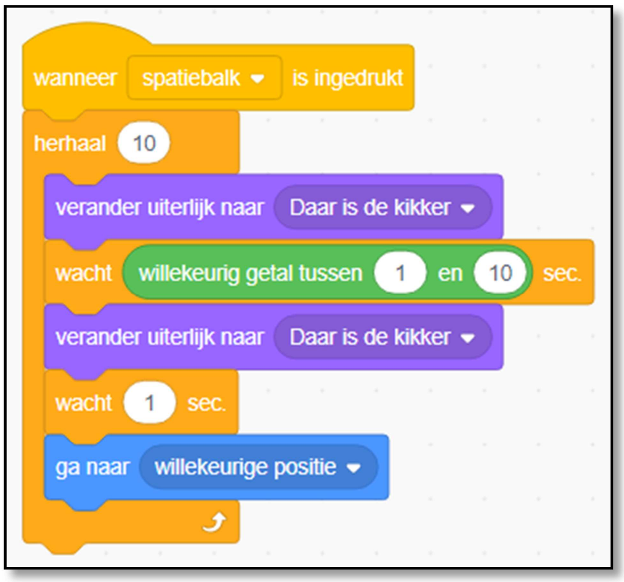


```
wanneer spatiebalk ▾ is ingedrukt
verschijn
ga naar Ruimteschip ▾
herhaal tot < <raak ik Muur ▾ ?> of <raak ik rand ▾ ?> >
    verander y met 10
    ????
verdwijn
```



```
A. maak score ▾ 5
B. verander score ▾ met 5
C. toon variabele score ▾
D. verberg variabele score ▾
```

**9. Je wilt dat de kikker tien keer verdwijnt en verschijnt als je op de spatiebalk drukt. Er is alleen een fout gemaakt. Wat is er verkeerd geprogrammeerd?**



```
wanneer spatiebalk ▾ is ingedrukt
herhaal 10
    verander uiterlijk naar Daar is de kikker ▾
    wacht (willekeurig getal tussen 1 en 10) sec.
    verander uiterlijk naar Daar is de kikker ▾
    wacht 1 sec.
    ga naar willekeurige positie ▾
```

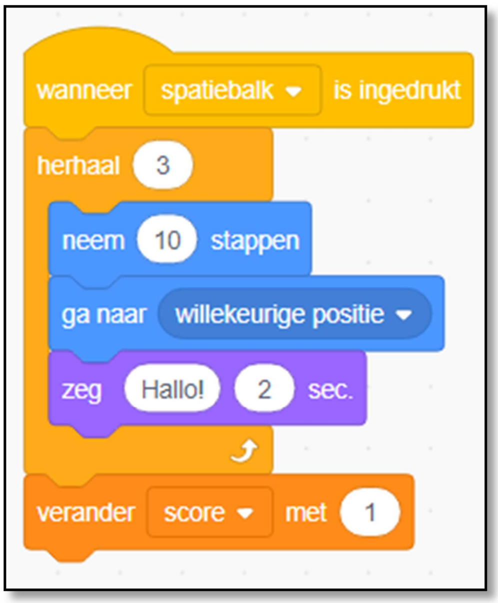
A. Verander uiterlijk naar ‘Daar is de kikker’ moet de 2e keer ‘Kikker is weg’ zijn.

B. De volgorde van de blokjes is verkeerd

C. Er zit geen fout in

D. Verander uiterlijk naar ‘Daar is de kikker’ moet de 1e keer ‘Kikker is weg’ zijn.

**10. Wat gebeurt er als je op de spatiebalk drukt?**



A.
- Je neemt tien stappen.
- Je gaat naar een willekeurige positie .
– Je zegt hallo 2 seconden lang.
- Je neemt tien stappen.
- Je gaat naar een willekeurige positie .
– Je zegt hallo 2 seconden lang.
- Je neemt tien stappen.
- Je gaat naar een willekeurige positie .
– Je zegt hallo 2 seconden lang.
- De score verandert met 1.

C.
- Je neemt tien stappen.
- Je gaat naar een willekeurige positie .
– Je zegt hallo 2 seconden lang.
- De score verandert met 1.

B.
- Je neemt tien stappen.
- Je gaat naar een willekeurige positie .
– Je zegt hallo 2 seconden lang.
- De score verandert met 1.
- Je neemt tien stappen.
- Je gaat naar een willekeurige positie .
– Je zegt hallo 2 seconden lang.
- De score verandert met 1.
- Je neemt tien stappen.
- Je gaat naar een willekeurige positie .
– Je zegt hallo 2 seconden lang.
- De score verandert met 1.

D.
- Je neemt tien stappen.
- Je gaat naar een willekeurige positie .
– Je zegt hallo 2 seconden lang.
- De score verandert met 1.
- Je neemt tien stappen.
- Je gaat naar een willekeurige positie .
– Je zegt hallo 2 seconden lang.
- Je neemt tien stappen.
- Je gaat naar een willekeurige positie .
– Je zegt hallo 2 seconden lang.
- De score verandert met 1.

**Appendix C.**

Questionnaire meeting 1 (MSLQ and demographic questionnaire)

**Voornaam**: …………………………………..

**Leeftijd**: …………………………………..

**Geslacht**: jongen meisje anders

**(1= helemaal niet waar voor mij tot 7 = helemaal waar voor mij)**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Ik denk dat de les leuk zal zijn | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Ik ben geïnteresseerd in programmeren | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Ik vind programmeren leuk | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Ik wil later programmeur worden | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Ik denk dat ik goede cijfers ga halen voor de programmeerles | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Als ik de stof niet begrijp vraag ik aan mijn klasgenoten of leraar of ze het willen uitleggen | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Ik weet dat ik mijn opdrachten en toetsen heel goed ga maken | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Ik vind de opdrachten die ik voor dit vak moet doen leuk | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Ik geloof dat ik de informatie uit de opdrachten kan begrijpen | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Ik maak alle opdrachten van dit vak omdat ik graag goed wil leren programmeren | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Ik weet zeker dat ik alles ga begrijpen | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Wanneer ik iets niet snap, ga ik op zoek naar hulp | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Ik denk dat ik voor dit vak wel een voldoende haal | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Ik wil heel graag begrijpen waar dit vak allemaal over gaat en de opdrachten helpen me daarbij. | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Als ik kijk naar wat ik moet doen, naar de leraar en naar wat ik al kan en weet, dan weet ik zeker dat ik het kan | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Ik vertrouw erop dat ik alles snap, ook als het wat moeilijker wordt. | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Ook door de opdrachten en het huiswerk, ben ik heel geïnteresseerd in waar dit vak over gaat. | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Als ik woorden niet begrijp, vraag ik aan de leraar die nog eens uit te leggen | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Ik weet zeker dat ik de vaardigheden die je bij dit vak leert, goed ga kunnen | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Door het maken van het huiswerk en de opdrachten leer ik niets extra's over dit vak. | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Heb je eerder geprogrammeerd? Ja Nee

Zo ja, Met welke programmeertaal?

Javascript Python Scratch HTML/CSS Lego Mindstorms Anders, vertel ons: ......................

**Wat doet een programmeur. Geef drie voorbeelden:**

..........................................................................................

..........................................................................................

..........................................................................................

## Appendix D.

NPV-J-2 scales Inadequacy (grey) and Tenacity (white) (Dutch)

| | | | |
|---|---|---|---|
| 1. Ik voel me meestal goed | Ja | ? | Nee |
| 2. Ik werk het liefst heel netjes | Ja | ? | Nee |
| 6. Er zijn maar weinig mensen die mij begrijpen | Ja | ? | Nee |
| 7. Ik werk meestal hard | Ja | ? | Nee |
| 11. Ik ben meestal zeker van mezelf | Ja | ? | Nee |
| 12. Ik werk vaak slordig | Ja | ? | Nee |
| 16. Ik ben vaak boos zonder dat ik weet waarom | Ja | ? | Nee |
| 17. Als ik ergens aan begin dan maak ik het ook af | Ja | ? | Nee |
| 21. Ik ben vaak erg verdrietig | Ja | ? | Nee |
| 22. Ik ben altijd op de afgesproken tijd thuis | Ja | ? | Nee |
| 26. Ik ben vaak moe | Ja | ? | Nee |
| 27. Ik ben een doorzetter | Ja | ? | Nee |
| 31. Ik ben vaak zenuwachtig | Ja | ? | Nee |
| 32. Ik ben meestal snel afgeleid | Ja | ? | Nee |
| 36. Ik denk vaak dat ik niets goed kan doen | Ja | ? | Nee |
| 37. Ik ga graag naar  school | Ja | ? | Nee |
| 41. Ik denk vaak dat ik niks waard ben | Ja | ? | Nee |
| 42. Ik gebruik mijn tijd goed | Ja | ? | Nee |
| 46. Ik denk vaak dat niemand van me houdt | Ja | ? | Nee |
| 47. Ik geef nooit op | Ja | ? | Nee |
| 51. Ik droom vaak over vervelende dingen | Ja | ? | Nee |

| | | | |
|---|---|---|---|
| 52. Ik doe meestal wat mij gevraagd wordt | Ja | ? | Nee |
| 56. Ik heb vaak een hekel aan mezelf | Ja | ? | Nee |
| 57. Ik doe wat mensen van mij verwachten | Ja | ? | Nee |
| 61. Ik heb vaak een slechte bui zonder dat ik weet waarom | Ja | ? | Nee |
| 62. Ik doe altijd goed mijn best | Ja | ? | Nee |
| 66. Ik ben vaak bang dat ik fouten ga maken | Ja | ? | Nee |
| 67. Ik doe altijd wat ik heb afgesproken | Ja | ? | Nee |
| 71. Ik heb vaak het gevoel dat alles me mislukt | Ja | ? | Nee |
| 72. Ik kan lang achter elkaar doorwerken | Ja | ? | Nee |
| 76. Ik maak me vaak zorgen over wat anderen van mij vinden | Ja | ? | Nee |
| 77. Ik vind dat je altijd je ouders moet gehoorzamen | Ja | ? | Nee |
| 81. Ik maak me vaak zorgen | Ja | ? | Nee |
| 82. Ik let goed op als er iets wordt uitgelegd in de klas | Ja | ? | Nee |
| 86. Ik voel me vaak eenzaam | Ja | ? | Nee |
| 87. Ik luister altijd goed naar volwassenen | Ja | ? | Nee |
| 91. Ik voel me vaak onzeker | Ja | ? | Nee |
| 92. Ik hou mijn spullen graag netjes in orde | Ja | ? | Nee |
| 96. Soms voel ik me zo slecht dat niemand iets goed kan doen | Ja | ? | Nee |
| 97. Ik doe de meeste dingen met plezier | Ja | ? | Nee |