Bachelor thesis
Computing Science

Radboud University

Introducing Programming in Primary Education: A Review of Scientific Literature

Author:
Abe Heemskerk
s4310659

First supervisor/assessor:
Prof. Dr. E. Barendsen (Erik)
Erik.Barendsen@ru.nl

Second assessor:
Dr. J.E.W. Smetsers (Sjaak)
s.smetsers@cs.ru.nl

October 13, 2020

Abstract

Everything around us is becoming more and more connected. Your car, your watch and even the coffee machine can be connected to the cloud. Therefore, it is pivotal to think about ways to teach our youngest generations the ins and outs on this subject. Already many studies have been conducted on the introduction of programming skills to young children. The aim of this thesis is to create an overview on what is known about teaching programming in primary education, i.e. children up to the age of 12 (k-6). To this end, a combination of scientific review studies and original studies were analyzed with a focus on specific learning goals, learning content, the student's understanding, instructional design and assessment. In conclusion it can be stated that a plugged project based intervention could help young students in becoming better programmers in the future!

Contents

Chapter 1

Introduction

Computers are becoming a greater part of everyone's lives (Wodjao, 2020). The so called "Internet of Things" is all around us. Everything is connected to each other, every system has a database in the cloud or operates digitally. In fact, it became an important aspect of our society so quickly that the foundation for teaching and understanding the basics of computer science were never thought of.

The youngest generation needs to come in contact with the "why" and the "how" questions surrounding computer science. Why is everything connected to each other? Why is my mobile phone connected to that specific access point? How is this connection established? And how is my data sent over this connection? If you never ask these questions a deeper understanding will never be achieved. Therefore, a true understanding is not present in the minds of future generations.

Wing (2006) states that not only computer scientists but everyone should have some understanding of computational concepts. A true understanding comes with understanding of the multiple levels of abstraction. Not only knowing how your computer is connected to the internet, but understanding why it is connected to the internet. The information technology (IT) sector is growing faster than ever before. The top three of fastest growing industries in the US are all IT related (Ibisworld, 2019). Gresnigt, Taconis, van Keulen, Gravemeijer, and Baartman (2014) also shows that many people work in jobs related to science and technology, and a great workforce with suitable schooling in these subjects is needed.

In other words, if we do not do anything our future will hold a shortage of people working in science and technology. We know that in primary school and in the teacher training program, there is a struggle to implement this kind of education. The importance of this subject has been acknowledged,

but an efficient way of implementation into the primary school curriculum has not yet been found.

A first step must be taken in integrating this into the standard curriculum. Learning how to program is the basis of computer science (Grover & Pea, 2013). Therefore, we take a look at what has been done when it comes to implementing programming education in primary schools. By implementing a programming curriculum, a first step will be taken in getting our younger generation ahead in the field of computer and information science.
It has been shown that most students develop their interest in and attitudes towards science before the age of 14 (Osborne & Dillon, 2008). Therefore, if a greater effort is put in ensuring that the quality of science education before this age is of the highest standards, a bigger interest in this specific field will be generated for the next generation.

When taking a look at how to introduce programming in the primary school curriculum, there are a lot of aspects that need to be taken into account. Simple examples are; what kind of teach material suits different classes? What can be taught (not to difficult)? What level of understanding do the children have? All these questions pop into mind when first asking the question: How to do integrate programming in primary schools.

The research will be contributing to the work that Shirley de Wit is doing together with Hanno van Keulen with the 'Tech your future' science group. They are researching the question 'What is the place of programming in elementary school?'. What has been tried in the field of introducing programming into the primary school curricula and how was this achieved at the time.

In the chapter Background an insight is given with regards to the theoretical background. In the chapter Research a overview is given of the studied case studies. Conclusion and Discussion both conclude and discuss the found results.

Chapter 2

Background

Computer programming is defined as follows:

"The activity or job of writing computer programs (Cambridge international dictionary of English, n.d.)."

This is a rather broad definition. Writing a computer program is mostly done via the use of a basic textual programming language (C, Java, etc..). These languages uses various syntax, libraries, a object oriented or imperative base for its language.

These definitions are too complex to teach in primary school (McCalla & Greer, 1993). Primary school has children ranging from ages K-6 (Kindergarten to 12 years old). Every age group and grade has a different curriculum. When defining a curriculum there are various aspects that should be taken into account. Van den Akker (2007) states that there are multiple questions that should be asked when you are creating a curriculum. (Van den Akker, 2007) defines the following questions:

- What do they need to learn?
- What are they learning?
- How are they learning?
- What is the teacher's role?
- What do they use to learn?
- With whom are they learning?
- Where are they learning?
- When are they learning?

- How are they being assessed?

These are a lot of questions, but they tackle the core principle when creating a curriculum. To define structure when asking these questions about how to integrate programming on primary schools, we apply the framework introduced by Magnusson, Krajcik, and Borko (1999). They had developed this framework with taking into account the specific pedagogical aspects.

"The figure can serve as a map for planning science teacher education experiences and for specifying desired knowledge outcomes of those experiences." (Magnusson et al., 1999)

The "figure" they are referring to, is the following: Goals and objectives, Students' understanding, Instructional strategies and Assessment. We use the framework that is provided by Magnusson et al. (1999) combined with the design questions of Van den Akker (2007).

2.1 Goals and objectives

When teaching a subject, whether it is programming or computational thinking, a clear scope needs to be defined. This way we know when a certain set goal has been achieved. When setting a goal, in the field of programming, there are different angles of approach you can take to obtain a goal, as stated by research done by Robins, Rountree, and Rountree (2003).

The first one is the comparison between an expert and a novice. In the definition of an expert or the definition of a novice a lot is left open to interpretation. Primary school cannot be defined as a single type or constant. It contains children from different ages: 4 to 12 years old. Therefore, there is a difference in skill level within this broad group. The first time someone comes into contact with a programming course they are, as of the definition, a novice in that specific area. But it is not the same 'level of novice' per grade. At every age a child has more life experiences and it is more developed than the year before. This is described in the way information can be presented to children in different age groups. Every grade is more experienced and can understand more than the grade below.

Bybee (2011) has come to that same conclusion, but with regards to the tools children can use. "In the early grades, students can learn to use appropriate instruments (e.g., rulers and thermometers) and their units in measurements and in quantitative results to compare proposed solutions to an engineering problem. In upper grades, students can use computers to analyze data sets and express the significance of data using statistics."

An example of this increasing difficulty are mathematics lessons on primary school. All throughout primary school we have had mathematics. Every year a couple of hours a week. Every year learning new things about the field of mathematics. Mathematics does not have a general book on how it should be taught. It is a carefully constructed curriculum to suit the capabilities of the grades, specified for each year. Teaching programming in K-6 must have the same framework.

van Bekkum (2017) integrates this problem in his proposed curriculum. He states that the described level of difference is essential in designing a curriculum. For each grade a different, higher and more difficult level of understanding of the grammar of programming. He states the following learning goals: algorithm, decomposition, patterns, repetition, errors, prerequistes, abstaction, function, variable and representation. (van Bekkum, 2017).

As long as the children do not notice the transition in subject, it is an indication that they are capable of performing this next step in understanding programming or a specified concept of computational thinking, and that a previous set goal has been achieved. Bybee (2011) refers to this as a standard experience.

After the comparison between an expert and a novice comes the comparison between comprehension and generation (Robins et al., 2003). When the information is taught, do they understand the holistic view of the basis of programming or are they only able to generate a logical answer to the presented problem question.

Let us emphasize this distinction of comprehension versus generation with a real life example. We can teach a child how to build a house using LEGO blocks. First we build a house with all sorts of blocks and it looks like a firm and steady house. When you ask a child to also build a house out of LEGO there are two types of approaches it may take. The first one is copy behaviour (generation). It sees the house you build and it starts to build the house according to the visuals of the house we have built. This way the foundation of building a house is not taken into consideration. The second way is building with a foundation, starting by building a base and building a house from the ground up (comprehension).

It is important that there is a understanding of how the house is built. When introducing a introductory programming course the same applies. It

is necessary that the children are able to generate an assessable product in the end, a product generated via comprehension instead of replication. If a teaching exercise is presented during class, children between K-12 are expected to generate a solution via comprehension. Planning and carrying out investigations to get a better understanding in solving a problem should be the standard experience in every k-12 classroom (Bybee, 2011).

Not only the representation and the framework are important criteria for the teaching programming for student k-6. Linn and Dalbey (1989) proposes a 'chain of cognitive accomplishments' of learning goals, that should arise from ideal computer programming instruction. This chain of accomplishments forms a good summary of what could be meant by deep learning in introductory programming. This chain is defined as follows:

- The first link is the features of the language being taught
- The second link is design skills
- The third link is problem-solving skills

This chain defines the way the cognitive accomplishments are introduced into the curriculum of the student.

Explanation of the chain of cognitive accomplishment: First, the student needs to understand the language the program uses. Without the understanding of the language or the usage of the tool a learning process cannot be enabled. This also applies to learning how to talk for example. If you do not know the words used in a specific language, it is nearly impossible to construct complete and correct sentences. This is the first link used by Linn and Dalbey (1989) and the second phase used by Zaharija, Mladenović, and Boljat (2013). Once the basic understanding of the language or tool has been established, the second link and third phase both state that the children/students now must learn how to use the language or tool to create a solution, i.e. understanding how multiple separate aspects can work together to create a coherent whole. After these stages, the children are able to use the language or tool properly. This brings us to the final and third link and fourth phase, executing the knowledge about the language or tool to a real life problem.

Zaharija et al. (2013) describes a more recent study that used a different version of this 'chain of cognitive accomplishments'. This study worked with four different phases, as compared to the three used by (Linn & Dalbey, 1989). It is a good example on why these steps are a solid basis for

each curriculum.

By using these steps, the underlying problem will be addressed and no result is obtained via trial-and-error (Zaharija et al., 2013).

When we look at Van den Akker (2007) the two questions that should be asked are: what do they need to learn? And what are they learning?

When the learning goal 'what do they need to learn?' has been established, smaller simpler goals must be set to achieve that goal. This is defined as the learning content which answers the question 'what are they learning?'.

The context of teaching and learning how to program on primary schools is a small scope. This is because children from ages k to 6 are not susceptible to understanding the more advanced terms and understandings that are connected to programming. You cannot start with explaining iteration, recursion or loops (McCalla & Greer, 1993). How to solve these obstacles, brings us to the pedagogical side of teaching a primary school.

2.2 Students' understanding

One of the most important aspects to consider when implementing a new course in the curriculum, are the students that are being taught. We are looking at introducing a programming course in Dutch primary schools. This contains children from the ages four or five to ages at twelve or thirteen. This is defined as K-6 (kindergarten to twelve years old). School is not only meant to teach children pure knowledge. Primary school is much more than a phase in which the children must absorb knowledge. It is the first time they come into contact with a lot of other things. They learn about making friends, what sport they like, they develop interest in activities and learn about what is right and wrong. Therefore, this is a difficult group to study.

Pointed out by Lye and Koh (2014) a better understanding is necessary of the students' engagement. The children on primary school are ranging from the ages 4 to 12. In all cases we know that concentration for a longer period of time is difficult for these children. Focusing on a task they are not interested in, is difficult as well. Therefore, the main task from a teachers' point of view is engaging the children in the task at hand (Duncan & Bell, 2015).

The engagement of the students combined with the not yet developed cog-

nitive capabilities of the children, results in the fact that complex problems cannot be presented to children. Since reducing a problem into smaller sub-problems is not possible for the children (Robins et al., 2003).

2.3 Instructional strategies

There are many different forms of education for children from k-6. Teaching programming has two ways in which it can be presented to the students: plugged and unplugged education. Plugged education is with the usage of a digital device whereas unplugged education is relying on tangible materials (Saxena, Lo, Hew, & Wong, 2020).

Other then plugged or unplugged programming, we could also take a look at what form of representation is used to present the problem. It can be presented as a visual or textual problem. The difference between visual and textual is what object is used to represent the problem that the student needs to solve. Visual representation is by means of an object, drawing or other non-textual visual image whereas the textual representation is purely based on representation via text.

When designing and presenting instructions, for students from k to 6, it is important to realise that without guidance on the cognitive aspects of computational practices and computational perspectives, the programming experience may be non-educative as students are not actively reflecting on their experience (Grover & Pea, 2013). They could be merely doing it in the trial-and-error mode rather than thinking as they are doing. This also refers to the section about comprehension as described in the goals and objectives chapter.

The implication described by Grover and Pea (2013) holds the same value for visual programming in k-6. These students need to work within a certain framework that is understandable from a cognitive point of view. A trial-and-error mode does not have the preference over a thinking-doing strategy. A thinking-doing strategy is also proposed by Lye and Koh (2014). Because of this implication, a framework must be kept simple (Grover & Pea, 2013). This way a focus on the task at hand is favoured. Bybee (2011) also suggests that a framework must be kept simple. Therefore, Bybee (2011) proposes that lower grades should different, more easy to use, tools than higher grades. Because the usage of a tool should not be an implication in the learning process.

From the questions that we asked based on Van den Akker (2007), we see that the most questions are applicable to the subsection instructional strategies, based on Magnusson et al. (1999).

- How are they learning?
- What is the teacher's role?
- What do they use to learn?
- With whom are they learning?
- Where are they learning?
- When are they learning?

Since these questions form the basis of what is being learned and how it is taught, many of the questions were expected to be answered in this section. This is due to the fact that, as said by Van den Akker (2007):

"The components of learning activities, teacher role, and materials & resources are at the core of the micro-curriculum in the classroom."

We are looking at introducing programming in primary schools. Thus, we already know the answers with regards to the questions; with whom are they learning? Where are they learning? When are the learning?
Our focus will therefore be on the questions raised by the core of the micro-curriculum. These components refer to the following questions:

1. How are they learning (learning activities)?
2. What is the teachers' role (teachers' role)?
3. What do they use to learn (materials and components)?

To answer the question how are they learning, we take a look at the statement made by Lye and Koh (2014). When considering a teaching method the researchers looked at the different intervention approaches. When trying to introduce computational thinking they described four methods:
"They are reinforcement of computational concepts, reflection, and information processing and constructing their own programs" (Lye & Koh, 2014).
The reflection method and the information processing method were only evident in higher education. Thus, not useful for studying students who attend primary school. This leaves the other two methods to be examined as answers on the question: how are they learning? Both those methods,

reinforcement of a concept and constructing their own program, can operate as a stand alone teaching method, but they can also co-exist. A form of reinforcement on a small program, created by the student, contains both methods as described by Lye and Koh (2014).

The second question we need to answer is: What is the teacher's role? A concluding argument about the participation and the functionality of the teacher in the class is not a point of focus for this review. However, the participation will be evaluated partly, because the way in which information is presented to the students will be analyzed later on in this review.

The last question we need to answer in the section of instructional strategies based on Van den Akker (2007) is: What do they use to learn? The main difference is what is being used as a tool when we look at plugged programming. These are different applications designed to let the students, k-9, focus on concrete physical motion (Duncan & Bell, 2015). The most common applications amongst case studies are: Logo, Scratch, Scartch Jnr., Dr. Scartch, Lego with the usage of CHERP and Game Maker Studio.

Since unplugged programming does not use computers at all. The learning process can be done, for example, with cards that contain instructions. Thus, creating a chain of actions that can be taken (Jeuring et al., 2016). It can also be done with every tangible object as long as that object can represent a chain of actions.

Answering the question with all the specific tools will be answered in the chapter containing the results.

2.4 Assessment

If we consider teaching and the education as a systematic approach, we can define teaching as any other system. It has a defined input. A process occurs over this input, which results in an output. Over this process we can check, assess, features of our system (Yüksel & Gündüz, 2017).

Assessment as we know it comes in two forms: formative assessment and summative assessment. This last form, summative assessment, is very common in the educational world (Yüksel & Gündüz, 2017). Summative assessment is defined as obtaining data to assess how much a student has learned at the completion of a course (Dixson & Worrell, 2016). Typical examples of summative assessment is an written exam, a final project or a final paper. It is an evaluation of a student at the end of a learning process against a benchmark of some kind.

Formative assessment is done by gathering data for improving the learning process of the student (Dixson & Worrell, 2016). Via monitoring the student's learning process and providing feedback and insights, the student can improve its own learning. Examples of formative assessment is any form where continuous feedback is given to the student: assessment for learning. Common forms are emphasizing a students strengths and weaknesses, talking about topics that are difficult for the student or summarizing lectures to check whether the student had understand the lecture.

In the educational system the most common form of assessment is summative. A final test at the end of a curriculum to test whether the student understands the learning content and has reached to learning goal. When it comes to an introductory programming course at a primary school a final test at the end of the course is not as common. As stated by Lye and Koh (2014) most of the research examines the programming process. The programming process was, in most cases, captured by observation. The researchers suggest that in the future a think-aloud method, where the student says what he thinks could be beneficial to better assess the process. A hindrance in the think-aloud method of assessment is that it does not work in a classical setting due to overlapping noises. This impairs the concentration level and, may, influence their thought processes. Therefore, a think-aloud method of assessment is only functional when assessed individually.

It has been established that formative assessment is important in the learning process of the student, but defining and integrating this form of assessment in the classroom has been shown to be a rather difficult task (Antoniou & James, 2014).

Important for any form of assessment is that all the different barriers that are present for each individual student must be taken into account when designing these assessments (Council et al., 2012). The student must be solely assessed if he or she reached the learning goal. Therefore, it cannot be too difficult to understand the assessment form or have any other obstructions. The last question we need to answer with regards to Van den Akker (2007) is: how are they being assessed? As defined above, this varies. It can be a summative final test or final project but formative assessment is more common compared to summative assessment. This is because most of the topics of programming are too difficult to teach to students who attend primary school. Recursion, iteration, complexity or writing an entire textual program is too hard. As said in the subsection about instructional implications, a student needs to understand the basics first.

Chapter 3

Aim of the study

The research question we are asking is the following:

What do we know about teaching programming in primary education?

The study will be based on a mix literature reviews and case studies. A overview of this study are will be created based on other reviews. A so called review of reviews. This way existing gaps in this study area can be identified (Gough, Oliver, & Thomas, 2017).

The theoretical aspects of the results will be based around four themes: goals, students, instruction and assessment. The research question will, therefore, be divided in four sub-questions. These questions are:

- What is known about teaching programming in primary education with regards to the goals and objectives?
- What is known about teaching programming in primary education with regards to the students' understanding?
- What is known about teaching programming in primary education with regards to the instructional strategies?
- What is known about teaching programming in primary education with regards to the assessment?

Chapter 4

Method

When evaluating a selection of articles to implement programming in elementary schools, there are multiple questions that come mind. There are many variables that should be taken into account when trying to figure out the best solution for this implementation. We have chosen to create a framework based on the variables states by Magnusson et al. (1999):

1. Goals and objectives
2. Students' understanding
3. Instructional strategies
4. Assessment

We took a look at what has been researched and what has been done, with regards to these four various aspects. When the articles have been analysed, we looked for different patterns that may or may not exists in these results. We tried to find a correlation between the articles and the case studies. What has been done to the extend in which there could be a conclusion and which aspects do still need further exploration when it comes to research.

4.1 Literature

A literature search was conducted to find relevant articles for this thesis. Academic search engine Google Scholar was used to collect and select studies.

4.2 Search terms

The used search terms were (computational thinking OR programming OR programming languages OR game making OR coding) AND (elementary school OR primary education OR middle school OR high school OR K-6

OR K-12) AND (plugged-in OR unplugged) AND (learning OR assessment OR teaching) AND (review OR case study).

Computational thinking was used as a search term since a lot of original studies, regarding programming, used a computational concepts as learning goal.

4.3 Selection criteria

Year of publication: only original empirical studies published in 2010 or later were included in this thesis.

Publication format: peer-reviewed articles published in scientific journals or as chapters in study books were included.

Content: included articles describe an original study or a review study of the introduction of programming or computational thinking in primary or secondary education, up to K-12. The study introduces a specific method for programming education, i.e. a tool, an online program, a game or a simulation.

Age: studies focusing on teaching programming for children up to 12th grade, were included for the (theoretical) background of this thesis. This involves children up to the age of 18. The focus of articles presenting specific original studies (described in the result section of this thesis) was on K-6 level education, displaying a population of children up to the age of 12.

Study design: the original studies present qualitative research with a focus on the development of computational thinking in a group of children of a specified age, in a specified environment, applying a specified learning task.

4.4 Article selection

Using the search terms and applying the selection criteria described above, 2550 hits were obtained. After deliberate consideration on source and relevance, a total of 40 articles (23 reviews and 17 case studies) were included in this study.

Chapter 5

Results

When going over the found results, the following was taken into account: first of, we only looked at original studies of recent years. The studies we were interested in, are the ones conducted from 2010 onwards (younger than 2010). This decision was made to ensure that the research performed was done with more recent technology. Since we wanted to sketch an image of the near past and the current state of the introduction of programming in primary schools, we only take original studies of which the studied group is k-6.

A total amount of 17 original studies were included in the results. This is after all the selection criteria were met. The results that have been found were concluded from research that has been conducted in various countries. The countries that were present in the data are: Greece, Scotland, Spain, Indonesia, Canada, United States of America, Germany, Chile and Turkey. Since we wanted to study the question; what has been done? It was important to include various tools and teaching methods in the results. The found results will be described in the following subsection with more details. Each subsection will try to analyse the results, with regards to the structure as we have defined throughout this paper by using Magnusson et al. (1999).

5.1 Goals and objectives

The goal and the objective is the initial start of each research (as defined in the chapter: Background). In this chapter we have also stated that a goal and objective is twofold. The goal and objective is divided in; the learning goal and the learning content.

The learning goal is in most studies the same. The goal that most studies want to achieve is to improve the thought process of the students by improving their understanding of computational concepts. Papadakis, Kalogiannakis, and Zaranis (2016), Wilson, Hainey, and Connolly (2013), Sáez-López,

Román-González, and Vázquez-Cano (2016), Jenson and Droumeva (2016), Moreno-León, Robles, and Román-González (2015), (Gregg, Tychonievich, Cohoon, & Hazelwood, 2012), Brackmann et al. (2017), Leifheit, Jabs, Ninaus, Moeller, and Ostermann (2018), del Olmo-Muñoz, Cózar-Gutiérrez, and González-Calero (2020), Tsarava et al. (2017) and Kazakoff, Sullivan, and Bers (2013). All state that their learning goal is to see if their teaching methods results in improving the students' understanding of computational concepts. There are different learning goals that teachers want the students to achieve. Understanding of computational concepts in taught via different concepts.

These concepts may vary. When the students are using a tool we can see different ways in which the students come into contact with these concepts. Examples are; the understanding and using of variables in solutions, understanding of basic operations of the learning tool, the understanding and usage of sequencing or the usage of conditionals in problem solving. However, in most cases there was no true defined computational thinking aspect that the students needed to achieve.

Other original studies focused more on improving the problem solving skills of the students (Fessakis, Gouli, & Mavroudi, 2013), (Pardamean, Evelin, & Honni, 2011) and (Kalelioglu & Gülbahar, 2014), i.e. learning how decomposition works, breaking down a bigger problem into smaller problems. The third, and final, learning goal that was established was to teach the understanding of the basic concepts of computer science or software engineering. This varies from creating a game (Förster, Förster, & Löwe, 2018) (Gutierrez et al., 2018) to programming a robot (Bers, Flannery, Kazakoff, & Sullivan, 2014).

	Amount of original studies
CT concepts	11
Problem Solving	3
Software Engineering	3

The complementary part to the learning goal was the learning content. Every study teaches in the use-modify-create method as described by Waite (2017). The studies can be divided in the actions they were performing to attain knowledge to achieve the learning goal. We can divide the learning content in one main class that contains three different sub-classes. The main class is the complete program. The learning content changes every hour, sessions, weeks to provide a complete introductory course (del Olmo-Muñoz et al., 2020) (Tsarava et al., 2017) (Kalelioglu & Gülbahar, 2014) (Bers et al.,

2014) (Kazakoff et al., 2013). In these complete programs every sub-class is present. The sub-classes, of the learning content, are:

- The student has to move an object via the use of commands from one place to the other. (Papadakis et al., 2016) (Fessakis et al., 2013) (Pardamean et al., 2011) (Brackmann et al., 2017)
- The student has to build an object or program. (Wilson et al., 2013) (Jenson & Droumeva, 2016) (Moreno-León et al., 2015) (Förster et al., 2018) (Kalelioglu & Gülbahar, 2014)
- The student is learning via a platform where to student can play with pre-existing applications. (Sáez-López et al., 2016) (Gregg et al., 2012) (Leifheit et al., 2018)

The ratio, between the different styles of learning content, is as follows:

	Amount of original studies
Complete Programs	5
Move	4
Build	5
Play	3

5.2 Students' understanding

The second aspect from (Magnusson et al., 1999) was the students' understanding. The common theme when studying primary school students is that there is a form of instructional implication. Trying to teach them a programming topic, or explain the problem at hand, can result in misunderstandings or different interpretations. These are to be divided into 3 sub-topics within instructional implications:

1. Cognitive skills
2. Pedagogical development
3. Difference between genders

The cognitive skills of children, on primary schools, is seen as an impairment when teaching programming. Introducing programming in the early grades is difficult. This is because understanding programming goes hand

in hand with understanding different levels of abstraction. We can tackle this problem by introducing problems for the children that they can relate to.

Zaharija et al. (2013) states that one of the difficulties was that children at that age have not yet developed abstract, hypothetical thinking. Because of that they can only solve concrete real world problems. Jonassen (2011) says that all teaching materials should be anchored in an authentic problem that is relevant to the learner. He does not specifically state that this is the case for children, but his theory applies to all types of students. Therefore, we can conclude that the children can relate to a physical, concrete problem.

This conclusion aligns with the thoughts of Piaget (1976) described in the context of k-8 by Council et al. (2007). During the Preoperational Stage (2 - 7 years), a beginning of understanding objects symbolically begins. This is followed by the Concrete Operational Stage (7 - 11 years), where the development of logical thought begins. A true understanding of abstract objects or abstraction in general is not developed until Formal Operational Stage (11 years and over). The difference in the teaching of abstraction to children attending primary school can be seen in the research done by Lindberg, Laine, and Haaranen (2019). The grades 1-3 are taught different topics of programming and abstraction than the grades 4-6.

To understand which level of steps can be taken, we take a look at what Bloom et al. (1956) tells us. He described the multiple levels of understanding that exist. It is called Bloom's taxonomy. The different levels are: remember, understand, apply, analyze, evaluate and create. This relates to the, earlier discussed, learning goals. Setting a realistic goal that levels with the cognitive understanding of the children is difficult when it is in preoperational stage or when it is in the concrete operational stage (Piaget, 1976). When teaching children it should be about the development of cognitive skills instead of pure code. The competence model applies to programming and should be taken into consideration when designing a curriculum: understanding, usage, communication and strategy (van Bekkum, 2017).

Cognitive skills are only mentioned in two of the seventeen original studies. Besides Zaharija et al. (2013), (Fessakis et al., 2013) mentions the cognitive skills of the students. They stated that a minor problem was the understanding of the software interface. The children could locate the ladybug and the leaf, but not yet understand the concept of navigation. Also was there a problem with orally expressing what they wanted ('Go up' - instead of turn left). The other studies did not mention any form of instructional implication because of a cognitive impairment. The second sub-topic was pedagogical development. Pedagogical development or pedagogical impairment has had no remarks within any of the original studies. A difference in

the degree of development between classmates can be an impairment. However, none of the studies showed this.

The defined implications are applicable to both genders. As stated by Jeuring et al. (2016) the fun boys and girls have in programming, in primary education, is almost equal. This translates to the fact that there is no difference in the ability the learn how to program between the two different genders. This research was done on the subject on programming on elementary school. So we assume that there is no significant difference in performance between boys and girls between the ages of 4 through 12. In the original studies it was not stated to be a central subject in the research. However, six studies have mentioned some relation between genders. The other studies did not mention the genders at all (except for the fact that boys and girls participated in the research).

Of the six studies that mentioned gender, five studies had an almost 50-50 percent participation rate of boys and girls (Moreno-León et al., 2015) (Förster et al., 2018) (del Olmo-Muñoz et al., 2020) (Gutierrez et al., 2018). Jenson and Droumeva (2016) did not mention how many of the 67 students were boys and how many were girls. Jenson and Droumeva (2016) mentioned that girls were a bit more shy than boys in the class and they had a lower active participation rate. They would not raise their hand as much and ask fewer questions to the teacher. Furthermore, non of the studies could see any significant difference in the performance or in the assessment of boys and girls (Moreno-León et al., 2015) (Förster et al., 2018) (del Olmo-Muñoz et al., 2020) (Gutierrez et al., 2018).

5.3 Instructional strategies

The questions we asked on instructional strategies, based on Van den Akker (2007) questions, were: What do they use to learn? How are they learning? What is the teachers' role?

The distribution is the following:

	Amount of original studies
Plugged	13
Unplugged	2
Both	2

Of the seventeen original studies, four used some form of unplugged method (Fessakis et al., 2013) (Brackmann et al., 2017) (Leifheit et al., 2018) (Tsarava

et al., 2017). All of these methods used simple pen and paper to let the students solve the problem at hand.

We have seen that fifteen original studies use some form of plugged teaching. The following tools are being used in the plugged programming studies:

	Amount of original studies
CHERP (Robotics)	2
Scratch	7
Logo	2
Different application	4

As we can see Scratch prevails. In this total the variations on the Scratch software are also included. This number includes the studies that use Dr. Scratch (Moreno-León et al., 2015) (Gutierrez et al., 2018) and the study that uses Scratch Jr. (Papadakis et al., 2016). The other studies that use Scratch are Wilson et al. (2013), Sáez-López et al. (2016),Förster et al. (2018) and Kalelioglu and Gülbahar (2014). Scratch is followed by Logo (Fessakis et al., 2013) (Pardamean et al., 2011) and CHERP that is being used with TabgibleK by Bers et al. (2014) and with LEGO Education WeDoTM Robotics Construction Sets by Kazakoff et al. (2013). The last studies all use a different application or website to teach the student. Jenson and Droumeva (2016) uses Game Maker Studio, Gregg et al. (2012) uses EcoSim, del Olmo-Muñoz et al. (2020) uses the website code.org and Tsarava et al. (2017) uses the application MIT AppInventor.

When we take a look at plugged teaching, we also seperated the usage of textual and/or visual programming. The applications were divided as follows:

	Amount of original studies
Visual	15
Textual	0

The following questions that must be answered are: How are they learning? And the question, what is the teachers' role? These questions will be answered in a combined answer. To separate the different ways of learning of how the teacher is involved, we take look at the possible ways to teach students:

- classical. This is with a teacher in front of the whiteboard or digiboard explaining the subject. A professional will explain the matter to the novice student.

- Project based. Students will be learning via the means of a project. The students must create the project. It will follow a manual or work towards a goal. At any point in time the student may ask a tutor or teacher for advice or help. This way the student will never get stuck.
- Mixed. A mixed combination where the teacher or tutor first teaches classical, followed by a project. These project are, most of the time, a group project.

When we took a look at the original studies, the following results were obtained:

	Amount of original studies
Classical	0
Project based	3
Mixed	14

Zero studies were conducted on the, traditional, classical way of teaching. Project based teaching was in the case of Sáez-López et al. (2016), Pardamean et al. (2011) and Jenson and Droumeva (2016). By far the most studies were a mix of both. In all the cases there first was an introductory classical part, followed by a project that the students had to do in pairs or groups. In all these cases there was a teacher or tutor present to answer possible questions (Papadakis et al., 2016) (Wilson et al., 2013) (Fessakis et al., 2013) (Jenson & Droumeva, 2016) (Gregg et al., 2012) (Brackmann et al., 2017) (Leifheit et al., 2018) (Förster et al., 2018) (del Olmo-Muñoz et al., 2020) (Tsarava et al., 2017) (Gutierrez et al., 2018) (Kalelioglu & Gülbahar, 2014) (Bers et al., 2014) (Kazakoff et al., 2013).

The duration of each of these interventions differs. The shortest intervention was the workshop given by Moreno-León et al. (2015) and the longest was the course given by Pardamean et al. (2011), which had a total of 16 session of 40 minutes of a 8 week period. All the other studies are between these lengths.

5.4 Assessment

The last segment of the Magnusson et al. (1999) subjects is the one about assessment. When it comes to teaching programming the most important part is that progress can be tested or monitored (van Bekkum, 2017). In the background we have described two ways in which assessment can take place, i.e. formative or summative. That is also the question that Van den Akker (2007) asked. How are they being assessed?

	Amount of original studies
Formative	1
Summative	16

The only formative assessment that was done, was done by Gregg et al. (2012). As stated: "The researchers did not perform formal assessments (e.g., tests or deadlinebased homework). However, they reviewed the students' work regularly and gave feedback often."(Gregg et al., 2012). This is a classical example of a formative assessment. The other sixteen studies all did some form of summative assessment. These assessment are divided in pre-test and post-test, assessment done via an observer or a combination both.

	Amount of original studies
Pre-post test	11
Observer	3
Combination	2

From the pre-post test studies there were studies that used regular questions as pre-post test (Fessakis et al., 2013) (Pardamean et al., 2011) (Leifheit et al., 2018) (del Olmo-Muñoz et al., 2020) (Kalelioglu & Gülbahar, 2014) (Kazakoff et al., 2013). Brackmann et al. (2017) used multiple-choice questions. There are two studies that used the application function as Dr. Scratch (Moreno-León et al., 2015) (Gutierrez et al., 2018). This application assigns a grade from 1-3 to certain computational thinking concepts based on your coding. Furthermore, Förster et al. (2018) assessed a project based on pre-established coding criteria and did Tsarava et al. (2017) use a reward system via badges.

The summative assessment via a general observer was done by Papadakis et al. (2016), Wilson et al. (2013) and Bers et al. (2014). A combination of these two methods was found in Wilson et al. (2013) and Jenson and Droumeva (2016).

Chapter 6

Conclusion & Discussion

Goals and objectives

Going over the results of this section the first thing that must be noticed is that the main focus of these studies lays on the goal of teaching computational concepts. What these concepts are varies per study. This can be a bit confusing since there is no clear definition of what "computation thinking concepts" are. For example, the ability to "solve a problem" can also be seen as a computational concept since problem solving has a focus on 'decompositional' thinking and sequence of actions (cause-effect). The same goes for the studies that focus on Software Engineering. These studies start with focusing on the understanding of abstraction, which can be seen as a computational thinking aspect as well.

Setting a goal needs to be clear. What the goal is may vary as long as the goal has some relationship with computational thinking. Computational thinking is the basis for teaching programming in the future, since textual programming languages are far too complex for students of this age to understand.

Students' understanding

Defining the sub-topics in the original studies was difficult. A lot of review studies did mention this. An explanation for why there was nothing or little to say about cognitive skills or pedagogical impairment can be given by the fact that the tools used are already created with some pedagogical impairments in mind. That is why the tools have big buttons, lots of colors and are easy to use, as the developers know what impairments students between the ages of four and twelve have. Furthermore, most of the lessons given are short and at most two times in one week. This may add to the children's motivation on the subject. However, as described in the studies, it is difficult to discuss the children's motivation since some of the courses were

based on students that entered on a voluntary basis. Thus, it was not part of their regular curriculum. Motives to participate in such programs were not further discussed.

In conclusion it can be stated that these aspects were not the main focus points of all the studies. It is important to keep the cognitive capabilities of the students in mind, as well as the pedagogical impairments. This is mostly already done by the tool that is being used and the teacher or school that is presenting or teaching this subject.

Instructional strategies

With regards to the instructional strategies it is pretty clear that the instruction and project must be a form of visual programming. Whether it is dragging blocks or clicking on buttons, it needs to be visual. This is based on the fact that all the found and tested tools used a visual programming interface. This can be derived of the basis that the textual concepts are too difficult. A plugged visual programming course will, as shown, have the advantage over alternative methods.

We can also conclude that the classical way of teaching is not as conventional as it used to be. None of the studies chose this way to teach. This makes sense, since the subject that is being taught is how to program. Learning how to program, learning how to build and create will be difficult to learn if the matter is only taught in the classical way. Interaction and create-modify has been found necessary.

Assessment

Summative assessment is the more traditional assessment, but we are trying to find a way to make assessing more formative. However, it is hard to find a way to replace this summative way of assessing. Assessing the performance and knowledge of a student against a pre-defined benchmark is still the easiest way. It will take a lot more effort to come up with an effective way to make formative assessment the standard. Formative assessment takes too much time and effort to assess all the individual students and provide them with solid feedback. Until this can be made more effective, there will not be any change coming soon.

When we assess all the reviews and the original studies we can see a clear trend. All these courses and studies focus on the precursor to textual programming: introducing some form of computational thinking. Various tools exist which are solely created to be used by primary school children for learning these concepts. They are proven to be successful in increasing the

understanding of the basic computational concepts. However, we need to keep in mind that all the review studies and original studies that have been studied, are studies that have been done outside of the Netherlands. Each country has its own culture and own ways of teaching. You cannot provide one single answer when it comes to introducing a new curriculum into an existing program. The same goes for introducing programming on primary schools. It has been shown that there are a lot of ways to introduce some type of introductory programming courses via the use of various e-tools and applications or introducing these concepts unplugged. This just needs to find its way into our present study program on the primary schools!

References

Antoniou, P., & James, M. (2014). Exploring formative assessment in primary school classrooms: Developing a framework of actions and strategies. *Educational Assessment, Evaluation and Accountability*, *26*(2), 153–176.

Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, *72*, 145–157.

Bloom, B. S., et al. (1956). Taxonomy of educational objectives. vol. 1: Cognitive domain. *New York: McKay*, *20*, 24.

Brackmann, C. P., Román-González, M., Robles, G., Moreno-León, J., Casali, A., & Barone, D. (2017). Development of computational thinking skills through unplugged activities in primary school. In *Proceedings of the 12th workshop on primary and secondary computing education* (pp. 65–72).

Bybee, R. W. (2011). Scientific and engineering practices in k-12 classrooms. *Science Teacher*, *78*(9), 34–40.

Cambridge international dictionary of English, p., year=1995. (n.d.). *Cambridge international dictionary of english*.

Council, N. R., et al. (2007). *Taking science to school: Learning and teaching science in grades k-8*. National Academies Press.

Council, N. R., et al. (2012). *A framework for k-12 science education: Practices, crosscutting concepts, and core ideas*. National Academies Press.

del Olmo-Muñoz, J., Cózar-Gutiérrez, R., & González-Calero, J. A. (2020). Computational thinking through unplugged activities in early years of primary education. *Computers & Education*, *150*, 103832.

Dixson, D. D., & Worrell, F. C. (2016). Formative and summative assessment in the classroom. *Theory into practice*, *55*(2), 153–159.

Duncan, C., & Bell, T. (2015). A pilot computer science and programming course for primary school students. In *Proceedings of the workshop in primary and secondary computing education* (pp. 39–48).

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education*, *63*, 87–97.

Förster, E.-C., Förster, K.-T., & Löwe, T. (2018). Teaching programming skills in primary school mathematics classes: An evaluation using game programming. In *2018 ieee global engineering education conference (educon)* (pp. 1504–1513).

Gough, D., Oliver, S., & Thomas, J. (2017). *An introduction to systematic reviews*. Sage.

Gregg, C., Tychonievich, L., Cohoon, J., & Hazelwood, K. (2012). Ecosim: a language and experience teaching parallel programming in elementary school. In *Proceedings of the 43rd acm technical symposium on computer science education* (pp. 51–56).

Gresnigt, R., Taconis, R., van Keulen, H., Gravemeijer, K., & Baartman, L. (2014). Promoting science and technology in primary education: a review of integrated curricula. *Studies in Science Education*, *50*(1), 47–84.

Grover, S., & Pea, R. (2013). Computational thinking in k–12: A review of the state of the field. *Educational researcher*, *42*(1), 38–43.

Gutierrez, F. J., Simmonds, J., Hitschfeld, N., Casanova, C., Sotomayor, C., & Peña-Araya, V. (2018). Assessing software development skills among k-6 learners in a project-based workshop with scratch. In *2018 ieee/acm 40th international conference on software engineering: Software engineering education and training (icse-seet)* (pp. 98–107).

Ibisworld. (2019). *Fastest growing industries in the us by revenue growth (%) in 2020.* `https://www.ibisworld.com/united-states/industry-trends/fastest-growing-industries/`.

Jenson, J., & Droumeva, M. (2016). Exploring media literacy and computational thinking: A game maker curriculum study. *Electronic Journal of e-Learning*, *14*(2), 111–121.

Jeuring, J., Corbalan, G., Montfort, J., Es, N., Leeuwestein, H., et al. (2016). *Vorm en effect van programmeeronderwijs in het primair onderwijs* (No. UU-CS-2016-005). UU BETA ICS Departement Informatica.

Jonassen, D. (2011). Supporting problem solving in pbl. *Interdisciplinary Journal of Problem-Based Learning*, *5*(2), 95–119.

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), 33–50.

Kazakoff, E. R., Sullivan, A., & Bers, M. U. (2013). The effect of a classroom-based intensive robotics and programming workshop on sequencing ability in early childhood. *Early Childhood Education Journal*, *41*(4), 245–255.

Leifheit, L., Jabs, J., Ninaus, M., Moeller, K., & Ostermann, K. (2018). Programming unplugged: An evaluation of game-based methods for teaching computational thinking in primary school. In *Ecgbl 2018 12th european conference on game-based learning* (p. 344).

Lindberg, R. S., Laine, T. H., & Haaranen, L. (2019). Gamifying program-

ming education in k-12: A review of programming curricula in seven countries and programming games. *British Journal of Educational Technology*, *50*(4), 1979–1995.

Linn, M. C., & Dalbey, J. (1989). Cognitive consequences of programming instruction. *Studying the novice programmer*, 57–81.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for k-12? *Computers in Human Behavior*, *41*, 51–61.

Magnusson, S., Krajcik, J., & Borko, H. (1999). Nature, sources, and development of pedagogical content knowledge for science teaching. In *Examining pedagogical content knowledge* (pp. 95–132). Springer.

McCalla, G. I., & Greer, J. E. (1993). Two and one-half approaches to helping novices learn recursion. In *Cognitive models and intelligent environments for learning programming* (pp. 185–197). Springer.

Moreno-León, J., Robles, G., & Román-González, M. (2015). Dr. scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia*(46), 1–23.

Osborne, J., & Dillon, J. (2008). *Science education in europe: Critical reflections* (Vol. 13). London: The Nuffield Foundation.

Papadakis, S., Kalogiannakis, M., & Zaranis, N. (2016). Developing fundamental programming concepts and computational thinking with scratchjr in preschool education: a case study. *International Journal of Mobile Learning and Organisation*, *10*(3), 187–202.

Pardamean, B., Evelin, E., & Honni, H. (2011). The effect of logo programming language for creativity and problem solving. In *Proceedings of the 10th wseas international conference on e-activities* (pp. 151–156).

Piaget, J. (1976). Piaget's theory. In *Piaget and his school* (pp. 11–23). Springer.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer science education*, *13*(2), 137–172.

Sáez-López, J.-M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "scratch" in five schools. *Computers & Education*, *97*, 129–141.

Saxena, A., Lo, C. K., Hew, K. F., & Wong, G. K. W. (2020). Designing unplugged and plugged activities to cultivate computational thinking: An exploratory study in early childhood education. *The Asia-Pacific Education Researcher*, *29*(1), 55–66.

Tsarava, K., Moeller, K., Pinkwart, N., Butz, M., Trautwein, U., & Ninaus, M. (2017). Training computational thinking: Game-based unplugged and plugged-in activities in primary school. In *European conference on games based learning* (pp. 687–695).

van Bekkum, W. (2017). Programmeren implementeren.

Van den Akker, J. (2007). Curriculum design research. *An introduction to educational design research*, *37*.

Waite, J. (2017). Pedagogy in teaching computer science in schools: a literature review (after the reboot: computing education in uk schools). *Online.(November 2017). Retrieved July*, *24*, 2019.

Wilson, A., Hainey, T., & Connolly, T. M. (2013). Using scratch with primary school children: an evaluation of games constructed to gauge understanding of programming concepts. *International Journal of Game-Based Learning (IJGBL)*, *3*(1), 93–109.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35.

Wodjao, T. B. (2020). A double-hurdle model of computer and internet use in american households. *Departement of Economics, Western Michigan University. Fabrizio Carlevaro, Yves Croissant, Stéphane Hoareau*, *49*.

Yüksel, H. S., & Gündüz, N. (2017). Formative and summative assessment in higher education: opinions and practices of instructors. *European Journal of Education Studies*.

Zaharija, G., Mladenović, S., & Boljat, I. (2013). Introducing basic programming concepts to elementary school children. *Procedia-social and behavioral sciences*, *106*, 1576.