Journal of the European Teacher Education Network
2017, Vol. 12, 13-24

Teaching Computational Thinking to Primary School Students via Unplugged Programming Lessons

Hylke H. Faber[1], Menno D. M. Wierdsma[1], Richard P. Doornbos[1]; Jan S. van der Ven[2] and Kevin de Vette[2]
[1]Hanze University of Applied Sciences; [2]Groningen Programmeert
Groningen, Netherlands

Abstract
This paper focuses on an introductory course in computational thinking for students at their final year in primary school, carried out at the start of the academic year 2015/2016. The course consisted of six 90 minutes' lessons that were taught once a week over the course of six weeks in 26 schools in the north of the Netherlands. The lessons were designed for students to study programming concepts without requiring computers or tablets. This paper describes the design and evaluation process for these 'unplugged' lessons in computational thinking. This paper ends with design principles for the design of lessons in computational thinking, and discusses possible directions for future research.

Key words: Computational thinking, programming, unplugged, K-12, primary education

Introduction

Many primary schools in the Netherlands are struggling with the way in which to use information technology in the classroom. While most primary schools have acquired at least a few computers or tablets for their students, many schools are uncertain of the way in which they should deploy these facilities. In order to solve this problem, the Netherlands Institute for Curriculum Development (SLO), responsible for designing the attainment targets for both primary and secondary education, has published a draft of a part of the future curriculum called *digitale geletterdheid* [digital literacy], which itself is part of the 21st century skills (SLO, 2015a). Computational thinking, together with basic IT skills, knowledge of new media and information processing skills, make up digital literacy. Closely related to programming, computational thinking differentiates itself from programming by being defined as a mental toolset (Selby & Woollard, 2014). Computational thinking includes ways of thinking and acting that can be applied to a multitude of real-world problem, stretching beyond programming.

Following the construction of a new data center in the north of Groningen, Google wanted to make a welcoming gesture to the local residents in the form of a programming course for students in their final year of primary school. An employee from Google would start the course and give an example of the real-world application of programming. The Hanze UAS was asked to provide introductory programming lessons. However, because the networking infrastructure was less than ideal in the rural area where most of the schools that would be receiving our lessons were located, we opted to create programming lessons that do not make use of the computer. Learning to program without a computer is known as 'unplugged' or 'offline' programming (Bell, Alexander, Freeman, & Grimley, 2009; Wohl, Porter, & Clinch, 2015).

Theoretical framework

Computational thinking is a collection of mental tools that enables the individual to solve problems more effectively by thinking like a computer scientist (Wing, 2006). The unclear exact definition of the term has caused some discord among academics, resulting in some researchers and organizations publishing their own definition (Barr & Stephenson, 2011; Brennan & Resnick, 2012; Selby & Woollard, 2014; Wing, 2010). Selby and Woollard (2014) published an interesting study in which they propose a definition of computational thinking based on the consensus between 39 different publications related to the definition of computational thinking. Their definition describes computational thinking as consisting of 7 aspects:

1. A thought process
2. Abstraction
3. Decomposition
4. Algorithmic design
5. Evaluation
6. Generalization
7. Automation

A thought process encompasses the notion that computational thinking is a mental toolset, frequently used to solve problems. All the other aspects of computational thinking describe a mental skill or way of thinking about either problems or their solutions.

Abstraction is the most essential and most distinctive aspect of computational thinking and sets it apart from other way of thinking (Wing, 2008). Abstraction lets an individual look at certain aspects of a problem or situation by hiding complex aspects of said problem or situation. By removing the aspects of a problem that are not relevant, the individual is not distracted and can direct all attention to the important aspects of the problem.

Decomposition is related to abstraction. It lets an individual tackle a complex problem by dividing it up in numerous small problems, based on functionality (National Research Council, 2011).

Algorithmic design is needed to structure the solution to the problem. Thinking algorithmically originated from computer science (National Research Council, 2010) and lets the individual create a strictly structured sequence of instructions. In some situations, an information-processing agent can be used to process the created algorithm. Some view the algorithm as the output of the process of computational thinking (Aho, 2012). This aspect of computational thinking lets the individual focus on the way in which a solution is structured. The sequences of the algorithm should leave as little room as possible for interpretation or uncertainty. By working in this way, the output of the algorithm can be precisely predicted and replicated.

Evaluation describes the process in which the effectivity and efficiency of the solution is assessed. Maybe the solution to the problem can be optimized to use fewer resources or be more time-efficient. This aspect of computational thinking tells the individual to search for the most effective or efficient solution to the problem.

Generalization challenges the found solution to the problem to be able to be applied to other and similar problems. Small chunks can be used to help solve a problem in other situations or can be improved based on the situation.

Automation lets the solution be carried out by an information-processing agent. This can be in the form of a computer that solves a complex calculation or repeats a monotonous task, or by using robotics to process the created algorithm. According to Wing (2008), a human can also take on the role of the computing agent.

In the Netherlands, the Netherlands Institute for Curriculum Development (SLO) has recently published their working definition of computational thinking (SLO, 2015b). At first glance, they seem to have copied and translated the definition used by the CSTA and ISTE (Barr & Stephenson, 2011). This specifies a more broadly defined working definition of computation thinking. In addition to the terms employed by Selby and Woollard, except evaluation and generalization, CSTA and ISTE introduce parallelization, simulation and three terms related to data handling. For more detail, see Barr & Stephenson (2011):

Data collection encompasses the skill to find and create a dataset that is relevant for solving the current problem.

Data analysis means analyzing the data which have previously been collected in such a way that meaningful conclusions can be drawn from them.

Data representation is putting the previously collected and analyzed data to good use by supporting the conclusions in either a graphical representation or a short summary of the most important points.

Parallelization means simultaneously processing a certain amount of data.

Simulation is used to gather data in a situation where real-world data would be impossible or impractical to collect.

Although Selby and Woollard's definition of computational thinking is more firmly rooted in academic habits, CSTA and ISTE's definition seems more widely adopted. Furthermore, the article by Barr and Stephenson gives a short summary of how computational thinking can be applied in situations other than computer science, such as mathematics, science, social studies and language arts (Barr & Stephenson, 2011). However, both definitions show a number of important similarities, such as the inclusion of abstraction, decomposition, algorithmic thinking or design and automation.

Many researchers agree that computational thinking is an important skill that should be taught to the next generation (Barr & Stephenson, 2011; Brown et al., 2013; Bundy, 2007; Grgurina, Barendsen, Zwaneveld, van de Grift, & Stoker, 2013; Grover & Pea, 2013; Hodhod, Khan, Kurt-Peker, & Ray, 2016; Kafai & Burke, 2013; Lu & Fletcher, 2009; Voogt, Fisser, Good, Mishra, & Yadav, 2015; Wing, 2006). In countries, such as the UK, Finland and the USA, computational thinking has been added to the national curriculum. However, the exact way in which computational thinking is being taught in these countries varies greatly. In the UK, for example, the Department for Education has added a number of set goals for each Key Stage (Department for Education, 2013a, 2013b).

These goals describe in detail what every student should be able to do or know in each specific Key Stage, which are age-bound phases of the British curriculum. Conversely, in Finland, teachers are encouraged to come up with their own implementation of computational thinking in their lessons. While they have more freedom to choose the way in which they want to teach the subject, some teachers feel uncertain on how they should approach this new subject.

Most researchers agree on the notion that teaching programming can be used to train computational thinking (Grover & Pea, 2013; Kafai & Burke, 2013; Lye & Koh, 2014; Mannila et al., 2014; Selby, 2014; Wang, Wang, & Liu, 2014). Further research needs to be done to reveal the relationship between programming and computational thinking.

Research question

The aforementioned project, in which six unplugged programming lessons aimed at teaching basic programming concepts to final year primary school students were created, plays a central role in this paper. The research question is as follows:

How can computational thinking be taught to final year primary school students without requiring the use of a computer?

In order to answer the research question, first the research methodology will be explained, as this will illustrate how the research question can be answered.

Research methodology

In order to find an answer to the research question, the research method of *educational design research* (EDR) was chosen (van den Akker, 1999; van den Akker, Bannan, Kelly, Nieveen, & Plomp, 2013). EDR focuses on creating and improving an educational design in order to gain more insight into the learning process for a particular topic and formulate principles for future educational designs regarding that topic. By collecting feedback from teachers after testing each revision of the design, the effectiveness of the design can be improved. In this study, we developed a design for a series of lesson materials aimed at teaching computational thinking. After using the lesson materials to teach computational thinking, feedback gathered from teachers was used to improve the lesson materials. In this preliminary explorative study, a prototype of the design was developed and evaluated. In a follow-up study, future revisions of the lesson materials will be evaluated and improved upon.

Analysis of the evaluation data can reveal design principles. These design principles can be used by others to create new lesson materials to teach computational thinking. The design principles state which elements of the lesson materials elicit positive reactions in both students and teachers. By improving each revision of the design, in a cyclic manner in which each design is tested and evaluated, more design principles can be acquired.

The design principles are the primary outcomes of this study. However, the lesson materials developed during the process are a secondary result outcome of this study. Both the lesson materials and the design principles are used to answer the research questions.

While design principles can have a more widespread effect, for instance when designing new lesson materials, the actual lesson materials developed during this study can be used directly by a primary school teacher to teach computational thinking.

The feedback needed to improve the design consists of focus group interviews with the teachers. Questions were related to the practical aspects of the lesson materials, such as the time needed to complete the lessons and what the reactions of the students were like. Next, we asked what the teachers liked and did not like about the lesson materials, and if they could come up with suggestions on how to improve the materials.

The created lessons were provided to 26 schools in the northern region of the province of Groningen in the Netherlands. In total, the lesson materials were used to teach 411 students, and we recruited 15 teachers to provide the lessons and give feedback during the focus group interviews.

Results

The results of this study are twofold. First, the unplugged lesson materials are concisely summarized below. Second, the design principles that have been extracted from the lesson materials are also presented.

Unplugged programming

The literature suggests that programming can be used to teach computational thinking, and that this can be achieved in an unplugged manner. We aimed to center each lesson around a single concept in the world of programming. The following concepts were chosen: algorithms, variables, repetition and conditionals. One of the lessons, focusing on the concept of algorithms, is based on a sample lesson we found online called *Robot taal* [Robot language] (Codekinderen, n.d.), originally developed by *Codekinderen* [Coding Kids], a project designed to introduce various programming concepts to primary school students without requiring the use of a computer. Furthermore, we chose to adapt another lesson by *Codekinderen* [Coding Kids], focused on binary counting, as we had learned from previous experiences that this lesson had elicited positive reactions from students. To wrap things up, we decided to create a lesson which would combine the concepts covered in previous lessons to create more complex programs.

Various online materials were used as an inspiration for the design of the lessons (Bell, Witten, & Fellows, 2006; Code.org, n.d.; Codekinderen, n.d.). Where applicable, the original authors of lesson materials gave their permission to use and adapt their content. All authors of this paper collaborated on creating the lesson materials. This resulted in combining the pedagogical and didactical content knowledge of the teacher trainers of the Hanze University of Applied Sciences with the programming background of Groningen Programmeert (Groningen Codes), a foundation aimed at teaching programming to primary school students. After agreeing on the programming concepts that would be covered in the course, we decided on the learning outcomes and activities of each of the lessons. Next, monthly collaborative design sessions were planned, focusing on refining various details of the lesson materials, such as how many assignments should focus on a given aspect of the central programming concept, or what metaphors would work best to explain a certain concept.

Lesson materials

Each lesson is structured using an on-screen presentation, a teacher guide, a set of assignments for the students and a copy of the assignments that have the correct answers already filled in. Each lesson starts with an explanation of a concept in coding and an example of how that concept is used to make everyday life easier and more efficient. This is usually followed by a classroom demonstration to introduce the central concept of the lesson. Afterwards, the students are asked to do group and individual assignments. Each lesson ends with a short summary, which includes short explanations of new words and phrases. Table 1 gives a concise overview of the created lessons, the related learning outcomes and the activities associated with the lessons.

Table 1
Overview of the lesson materials

	Concept	Learning outcomes	Activities
1	Binary counting	Students can convert decimal numbers to binary and back and explain the relationship between letters, decimal and binary numbers.	Use paper cutouts to represent binary numbers. Decipher various binary codes. Create a binary code to communicate a word to a classmate.
2	Algorithms	Students understand that a computer cannot think and needs to be programmed in a very precise way in order to get it to do what you want, and experience the process of debugging.	Take on the role of the programmer, who writes an algorithm for the robot. Take on the role of the cup robot, which reads an instruction to create a certain arrangement of cups.
3	Variables	Students can explain how variables are used in real-life situations, can name three properties of variables and can name three types of variables.	Try to come up with possible variables based on give real-world examples. Design a passport for an imaginary creature.
4	Repetition	Students can shorten a long list of similar instructions by introducing repetition.	Find out a quick way to play the game ‘guess my number’. Draw a number of lines in a circle using only a few lines of code.
5	Conditionals	Students understand how a program can react to the value of a variable and can predict the outcome of program using conditionals.	Play a simple game with a deck of playing cards in which points are awarded based on certain aspects of the drawn playing card.
6	Combining previously learned concepts	Students can combine various programming concepts and understand that the order of various instructions can influence the number of processing steps required.	Sort classmates based on gender, date of birth and number of letters in their name. Sort a given collection of animals based on various aspects.

Lesson 1 focuses on binary counting. While this has nothing to do with programming at this stage, it forces the students to take a different perspective on numbers. They learn to see numbers as simple symbols that represent mathematical values, by converting binary numbers to decimal numbers and back. Later on, in the lesson, students are challenged to convert numbers to letters. This leads to a better understanding of the computer keyboard, where each key has a unique binary value associated with it.

Lesson 2 introduces the concept of algorithms by letting the student engage in taking the role of a robot. While working in pairs, students have to write a program for the robot to process. The goal is to let the robot construct a small structure made out of plastic cups. They are allowed to choose from 6 different commands to create a program that will ultimately result in the desired arrangement of cups. By carefully selecting the necessary steps, the students learn how to create an algorithm.

Lesson 3 teaches the students the concept of variables. Variables are introduced as a way to keep score during a game. Variables consist of a name and a value. The variable's name stays the same, but the value changes as the game goes on. Several more examples of variable in real-life situations are given, after which students are asked to come up with variables themselves. Students are then encouraged to come up with a passport of an imaginary animal and have to answer questions related to variables from the passport they created. This also introduces three different types of variables: text-variables, number-variables and booleans. Booleans are explained more thoroughly as the concept of true/false can be challenging. At the end of the lesson, a demonstration is given on how a real computer program deals with variables and how the type of a variable can have unexpected results when combining different variables.

Lesson 4 deals with the concept of iteration, processing the same instructions multiple times, while sometimes only changing one aspect of the instruction. By programming this way, the students are shown they can sometimes shorten the list of instructions by using repetition. Students are invited to think of situation where iteration could be useful. At the end of the lesson, a quick sorting algorithm using iteration is used to sort 8 students based on a number they received beforehand. With only a few lines of code, the 8 students are neatly sorted. It does not matter how 'unsorted' the starting positions are, with a simple program the result is always the same. This is a useful demonstration of the power of iteration.

Lesson 5 shows students that computer programs can respond to their environment by making use of conditions. Only when the conditions of a certain situation are met, the instructions associated with that condition are executed. In this lesson, students are taught to write down actual computer code. The concept of conditional statements is introduced by playing a simple game using playing cards. Students are given points for the aspects of each card they draw. For instance, they get a different number of points based on the suit of the card. Students are encouraged to translate the rules of the game into computer code.

The final lesson, lesson 6, combines all the concepts of programming they have previously learned. The concepts of algorithms, variables, iteration and conditions are re-introduced and combined to create a more complex program. Students are invited to sort their own classmates based on certain characteristics, like gender, number of letters in their name and date of birth.

Design principles

Feedback from the lessons was acquired during three one-hour long focus group interview sessions. This revealed some interesting evaluation data. Feedback related to aspects of the lesson materials that are not relevant for future research, such as time needed to complete the lesson, or the discovery of spelling mistakes in the guide- or workbooks, are not summarized here.

The unplugged aspect of the lesson was met with enthusiasm. Most course teachers appreciated the unplugged aspect of the lesson materials, whereas some of the students were hoping to learn how to hack their favorite online Flash-based game. During later lessons, these students eventually came to appreciate the hands-on experiences they received. Even some of the regular teachers, who had little to no programming experience, were pleasantly surprised by the way in which the students were actively engaged using the lesson materials and could work in a creative way during the course. Most students appreciated the way in which new concepts were explained by playing a small and simple game. For instance, when introducing conditional statements, students were allowed to play a game in which they would each take a card from a deck of playing cards. The number of points they would get was dependent on the suit of the card. It is possible that playing a game while simultaneously learning a new programming concept can improve motivation in these students.

Many teachers reported a significant difference in skill level within the classroom. Some students were eager to work on bigger challenges, while some others had great difficulty to grasp the concept of that particular lesson. We would recommend, at least for unplugged, but probably programming lesson courses for primary education as a whole, to be aware of this range of skill levels and to adapt lesson materials in order to be able to provide engaging lesson time for each student.

Some teachers reported students had difficulty remembering all the new words they learned during the lessons. In order to solve this, we added a new PowerPoint sheet to the presentation in which we would summarize all the new words that would come up during the lesson, with a short explanation on what they mean or represent.

In order to help the students, understand how each programming concept is used in the real world, we added a number of sheets to the presentation at the beginning of each lesson. Some students experienced difficulty in seeing the purpose and advantages of using certain programming concepts. Sheets with pictures and examples are now part of the PowerPoint presentations for each lesson in which a new concept in introduced.

Of the programming concepts introduced, variables were by far the most difficult to teach, according to some of the teachers. In order to mitigate this problem, the lesson materials for this concept received a major overhaul. Instead of using cups as a metaphor for variables, with a sticker on the side with the name of the variable written on it, and a number of counters or a piece of paper inside the cup as the value of the variable, we opted for a different approach. We now introduce the concept as something that can change between individual instances of the same object, such as the number of petals on a flower, or the price of a pair of jeans. Next, we specify the difference between a text-variable and a number-variable. The name of the variable, as something that does not change, is not introduced during the lessons. Even though it stands in contrast with the value of a variable, as something that does change, some teachers reported a lot of confusion during the lessons.

Lastly, some teachers reported they made use of a graphical programming environment to let some of the students, who had finished with the regular assignments, explore more challenges. The addition of some type of online assignment elicited a lot of positive reactions from the students. These online assignments challenged the students to use a graphical programming environment.

Discussion

Even though this paper presents a small number of design principles for teaching programming to primary school students, more work still needs to be done. These preliminary results need to be verified and expanded.

The results presented in this paper were distilled from only a small number of short interviews. We believe this is not a problem for a mere explorative study. However, future research should expand these design principles and focus on more systematic evaluation. We think the design principles presented in this paper provide a fertile foundation for further research, as they can be used to design other lesson materials to teach programming.

Even though this paper lacks any statistical analysis, the unplugged aspect of the lesson materials seems to elicit positive reactions from both teachers and students. We believe unplugged programming lessons are a valuable alternative to regular, online programming lessons.

The feedback gathered from the evaluation sessions was used to improve the lesson materials. Along with an updated design and new presentations, the new lesson materials are freely available for download at www.hanze.nl/programmeren.

Upcoming research

In the academic year 2016/2017 a new project with even more schools in the city of Groningen is starting. This pilot will hopefully result in more interest for education in computational thinking. During this 3-year project we will once again create programming lessons for primary school students, this time starting with five-year-olds and continuing into secondary education. This way, students will experience computational thinking through all of primary school, which allows secondary education to further build upon the skills learned in primary school.

References

Aho, A. V. (2012). Computation and Computational Thinking. *The Computer Journal*, *55*(7), 832–835. https://doi.org/10.1093/comjnl/bxs074

Barr, V., & Stephenson, C. (2011). Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Education Community ? *ACM Inroads*, *2*(1), 48–54. https://doi.org/10.1145/1929887.1929905

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer Science Unplugged:

School Students Doing Real Computing Without Computers. *Journal of Applied Computing and Information Technology*, *13*(1), 20–29.

Bell, T., Witten, I. H., & Fellows, M. (2006). *CS Unplugged.*

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association* (pp. 1–25). Retrieved from http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

Brown, N. C. C., Kölling, M., Crick, T., Peyton Jones, S., Humphreys, S., & Sentance, S. (2013). Bringing computer science back into schools. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (pp. 269–274). New York, New York, USA: ACM Press. https://doi.org/10.1145/2445196.2445277

Bundy, A. (2007). Computational Thinking is Pervasive. *Journal of Scientific and Practical Computing*, *1*(2), 67–69.

Code.org. (n.d.). Conditionals. Retrieved October 2, 2015, from https://studio.code.org/s/course2/stage/12/puzzle/1

Codekinderen. (n.d.). Unplugged. Retrieved from http://www.codekinderen.nl/leerling/unplugged/index.html

Department for Education. (2013a). Computing Programmes of Study : Key Stages 1 and 2, (September 2013), 1–2. Retrieved from https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/239033/PRIMARY_national_curriculum_-_Computing.pdf

Department for Education. (2013b). Computing Programmes of Study : Key Stages 3 and 4, (September 2013), 1–2. Retrieved from https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/239067/SECONDARY_national_curriculum_-_Computing.pdf

Grgurina, N., Barendsen, E., Zwaneveld, B., van de Grift, W., & Stoker, I. (2013). Computational thinking skills in Dutch secondary education. In *Proceedings of the 8th Workshop in Primary and Secondary Computing Education* (pp. 31–32). New York, New York, USA: ACM Press. https://doi.org/10.1145/2532748.2532768

Grover, S., & Pea, R. D. (2013). Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher*, *42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Hodhod, R., Khan, S., Kurt-Peker, Y., & Ray, L. (2016). Training Teachers to Integrate Computational Thinking into K-12 Teaching. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 156–157). New York, New York, USA: ACM Press. https://doi.org/10.1145/2839509.2844675

Kafai, Y. B., & Burke, Q. (2013). Computer Programming Goes Back To School. *Phi Delta Kappan*, *95*(1), 61–65.

Lu, J. J., & Fletcher, G. H. L. (2009). Thinking about computational thinking. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education* (pp. 260–264). New York, New York, USA: ACM Press. https://doi.org/10.1145/1539024.1508959

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. https://doi.org/10.1016/j.chb.2014.09.012

Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., & Settle, A. (2014). Computational Thinking in K-9 Education. In *Proceedings of the Working Group Reports of the 2014 on Innovation 38; Technology in Computer Science Education Conference* (pp. 1–29). New York, New York, USA: ACM Press. https://doi.org/10.1145/2713609.2713610

National Research Council. (2010). *Report of a Workshop on The Scope and Nature of Computational Thinking. Report of a Workshop on The Scope and Nature of Computational Thinking*. Washington, D.C.: The National Academies Press. https://doi.org/10.17226/12840

National Research Council. (2011). *Report of a Workshop of Pedagogical Aspects of Computational Thinking*. Washington, D.C.: The National Academies Press. https://doi.org/978-0-309-21474-2

Selby, C. C. (2014). *How Can the Teaching of Programming Be Used to Enhance Computational Thinking Skills?*

Selby, C. C., & Woollard, J. (2014). *Refining an Understanding of Computational Thinking*.

SLO. (2015a). 21e eeuwse vaardigheden. Retrieved February 1, 2016, from http://curriculumvandetoekomst.slo.nl/21e-eeuwse-vaardigheden/

SLO. (2015b). Een voorbeeldmatig leerplankader. Retrieved July 14, 2016, from http://curriculumvandetoekomst.slo.nl/21e-eeuwse-vaardigheden/digitale-geletterdheid/computational-thinking/voorbeeldmatig-leerplankader

van den Akker, J. (1999). Principles and Methods of Development Research. In J. Van den Akker, R. M. Branch, K. Gustafson, N. Nieveen, & T. Plomp (Eds.), *Design Approaches and Tools in Education and Training* (pp. 1–14). Dordrecht: Springer Science+Business Media. https://doi.org/10.1007/978-94-011-4255-7

van den Akker, J., Bannan, B., Kelly, A. E., Nieveen, N., & Plomp, T. (2013). Educational Design Research, Part A: An introduction, 206. https://doi.org/10.1007/978-1-4614-3185-5_11

Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, 715–728. https://doi.org/10.1007/s10639-015-9412-6

Wang, D., Wang, T., & Liu, Z. (2014). A Tangible Programming Tool for Children to Cultivate Computational Thinking. *The Scientific World Journal*, *2014*, 1–10. https://doi.org/10.1155/2014/428080

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions. Series A, Mathematical, Physical, and Engineering Sciences*, *366*(1881), 3717–3725. https://doi.org/10.1098/rsta.2008.0118

Wing, J. M. (2010). Computational Thinking: What and Why? *TheLink - The Magaizne of the Varnegie Mellon University School of Computer Science - The Magaizne of the Varnegie Mellon University School of Computer Science*, (2010), 1–6. Retrieved from http://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Wohl, B., Porter, B., & Clinch, S. (2015). Teaching Computer Science to 5-7 year-olds: An initial study with Scratch, Cubelets and unplugged computing. In *Proceedings of the Workshop in Primary and Secondary Computing Education* (pp. 55–60). New York, New York, USA: ACM Press. https://doi.org/10.1145/2818314:2818340