Informatics in Education, 2011, Vol. 10, No. 1, 73–88

Teaching Programming in Secondary School: A Pedagogical Content Knowledge Perspective

Mara SAELI[1], Jacob PERRENET[1], Wim M.G. JOCHEMS[1], Bert ZWANEVELD[2]

[1] *Eindhoven University of Technology, Eindhoven School of Education*
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
[2] *Open Universiteit Nederland, Ruud de Moor Centrum*
P.O. Box 2960, 6401 DL Heerlen, The Netherlands
e-mail: m.saeli@tue.nl, j.c.perrenet@tue.nl, wim.jochems@esoe.nl, bert.zwaneveld@ou.nl

Received: December 2010

Abstract. The goal of this literature study is to give some preliminary answers to the questions that aim to uncover the Pedagogical Content Knowledge (PCK) of Informatics Education, with focus on Programming. PCK has been defined as the knowledge that allows teachers to transform their knowledge of the subject into something accessible for their students. The core questions to uncover this knowledge are: what are the reasons to teach programming; what are the concepts we need to teach programming; what are the most common difficulties/misconceptions students encounter while learning to program; and how to teach this topic. Some of the answers found are, respectively: enhancing students' problem solving skills; programming knowledge and programming strategies; general problems of orientation; and possible ideal chains for learning computer programming. Because answers to the four questions are in a way not connected with each other, PCK being an unexplored field in Informatics Education, we need research based efforts to study this field.

Keywords: computer science education, informatics, pedagogical content knowledge, secondary education, programming, teaching.

1. Introduction

The 21st century is characterized by the ubiquitous presence of technology in everyday life. New generation students are surrounded by computer related instruments and will possibly do a job that has not been invented yet. Computing succeeded to conquer most of the aspects of our society and, in order to fit in, people need to be versatile and adaptable to modern and future technology. This scenario emphasizes the need to provide an education that can offer students and future adults the ability to understand and work with computer related instruments. The aim of Computer Science Education Research (CSER) is to improve the quality of the teaching and learning of the topics relative to this computerized world. A review of the literature (Holmboe *et al.*, 2001) evidences the existing need to broaden the efforts of informatics educators to contribute to the knowledge of why informatics should be taught at all, how informatics should be taught, what topics of informatics should be taught, and for whom the teaching of informatics is meant. In this

literature review we are particularly interested in answering these questions relative to a specific topic of informatics: Programming.

In this article we refer with the term of informatics as the computer science education delivered to upper secondary school students (14 to 18 years old). There seems to be no distinction in the use of these two terms by the CSER community.

The answers to the four questions introduced lead to the understanding of the concept called Pedagogical Content Knowledge (PCK; Shulman, 1986). PCK is a concept that combines the knowledge of the content (e.g., maths, informatics, etc.) to the knowledge of the pedagogy (e.g., how to teach maths, how to teach informatics, etc.), giving insights into educational matters relative to the learning and teaching of a topic. Teachers with good PCK are teachers who can transform their knowledge of the subject into something accessible for the learners. Studies portraying the PCK of Programming will enable informatics teacher trainings to improve their programs (Lapidot and Hazzan, 2003), boosting in this way their future teaching. There is evidence of the international interest (Stephenson *et al.*, 2005; Ragonis *et al.*, 2010; Woollard, 2005) on this topic, where the first efforts have been made to achieve the goal to portray the PCK of informatics.

PCK is a construct specific to teachers' knowledge; therefore teachers are the focus of this article. There are other aspects of the teaching and learning of programming that are as important as teachers' knowledge. Examples could be gender issues, mostly dealing with motivations that bring boys and girls to enrol in informatics courses; and students' motivation, which is part of general pedagogical knowledge. However, these topics of interest, despite very important, are not the focus of this paper.

2. Programming Education

Programming is only one of the topics concerning the teaching of informatics. In the Netherlands, informatics has been defined as a new generation discipline, because it is linked with Mathematics, Physics, Engineering, Linguistics, Philosophy, Psychology, Economy, Business, and Social Sciences in general (Mulder, 2002). If on one hand this complexity results in a relatively difficult job for researchers in this field, on the other it is possible to rely on the research achievements already obtained in the above mentioned disciplines. As Guzdial suggests, the basic mechanisms of human learning haven't changed in the last 50 years (2004) and we can prevent the reinvention of the wheel by looking at research in education, cognitive science and learning sciences research (Almstrum *et al.*, 2005).

A popular definition is that programming is the process of writing, testing, debugging/troubleshooting, and maintaining the source of code of computer programs (Wikipedia, 2007). We will later see that programming is a much broader topic than that described by the latter definition, as for example the ability to solve a complex problem with a top-down approach. Programming is a skill that is considered hard to learn and even after two years of instruction, the level of programming understanding is low (Kurland *et al.*, 1989). However, if supported by suitable teaching strategies and tools it can be mastered by pupils to some extent (Papert, 1980).

In this literature study we refer to programming as the topic used to introduce upper secondary school students to computer programming. We will not refer to specific Programming Languages (e.g., Java, Python, etc.), because we consider these as a mean/tool to achieve the teaching of Programming. Secondary school students should be taught programming concepts independent of specific applications and programming languages (Stephenson *et al.*, 2005; Szlávi and Zsakó, 2006).

3. Pedagogical Content Knowledge

Pedagogical Content Knowledge (PCK), a concept introduced by Shulman (1986, 1987), is defined as:

> The ways of representing and formulating the subject that make it comprehensible to others (Shulman, 1986, p. 9).

There is in fact a difference between knowing how to program and being able to teach programming. The classroom, where learning and teaching occur, is a complex environment in which several processes and actions happen. But when talking about PCK a special attention should be spent on students' learning. An aspect of PCK concerns the need teachers have to represent and formulate the subject, so that comprehension can occur. From the literature we know that different learners have different learning styles (Rayner and Riding, 1997), and needs. This implies that:

> [. . .] there are no single most powerful forms of representation, the teacher must have at hand a veritable armamentarium of alternative forms of representation, some of which derive from research whereas others originate in the wisdom of practice. Pedagogical content knowledge also includes an understanding of what makes the learning of specific topics easy or difficult: the conceptions and preconceptions that students of different ages and backgrounds bring with them to the learning of those most frequently taught topics and lessons. If those preconceptions are misconceptions, which they so often are, teachers need knowledge of the strategies most likely to be fruitful in reorganizing the understanding of learners, because those learners are unlikely to appear before them as blank slates (Shulman, 1986, p. 9).

An example in informatics could be the teachers' knowledge about the concept of programming structures, and the need to formulate their knowledge in a way that can be easily understood by their students. All research in this domain agrees on claiming that PCK is a knowledge that develops with years of teaching experience (Rovegno, 1992; Grossman and Lynn, 1990; Loughran *et al.*, 2001; Morine-Deshimer and Kent, 1999; Van Driel *et al.*, 1998; Sanders *et al.*, 1993), because teachers need to build up "a veritable armamentarium" of representations (Shulman, 1986).

The concept of PCK has been largely assimilated in educational research (Carpenter *et al.*, 1988; Cochran *et al.*, 1993; Van Driel *et al.*, 1998; Peterson *et al.*, 1989; Rich, 1993; Rovegno, 1992; Sanders *et al.*, 1993) and some scholars have reformulated it (Grossman, 1989, 1990; Hashweh, 2005; Marks, 1990; An *et al.*, 2004; Turner-Bisset, 1999). A deep and broad PCK is important and necessary for effective teaching (An *et al.*, 2004, Magnusson *et al.*, 1999). Moreover, Hashweh (2005) underlines how the teacher's approach

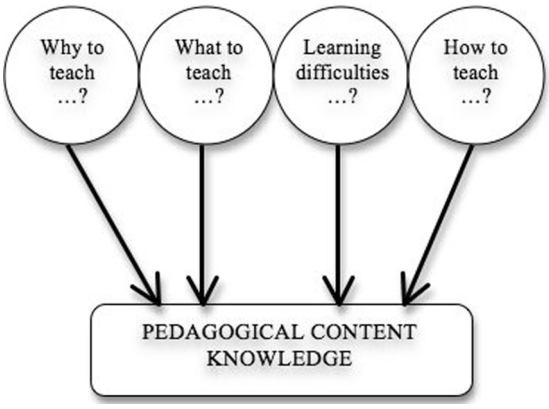

Fig. 1. Diagram based on Grossman's reformulation of PCK.

or orientation to his or her discipline (personal beliefs) influences the teaching of a certain topic, and might influence her/his PCK. This means that each teacher's PCK is in a way personal.

For the purpose of this literature study we will use the reformulation of the concept of PCK proposed by Grossman (1989, 1990). We choose her reformulation because we think that it schematizes the PCK through simple and easy to use questions such as: why teach a certain subject?; what should be taught?; and what are learning difficulties?. In our study we add a fourth question, which refers to the teaching methodology: how to teach?. This last aspect has been also lately introduced by Grossman (1990). We think that the latter is an important aspect of the PCK of a topic because it gives an insight of what teachers actually do. The question used to uncover this aspect of PCK is: how should the topic be taught?. By answering these four questions (see Fig. 1) it will be possible to define the PCK of a certain subject. The four questions are all connected with each other, because the reasons to teach a topic (first question) will influence the contents chosen to be included in the curriculum. In addition the learning difficulties that students encounter will surely influence the way to teach it.

4. Methods and Aims

This literature study has been conducted by exploring the existent literature available. Sources include printed articles, books chapters and information found on the Internet. The research has been conducted by using searching engines such as Google Scholar, or by browsing references lists of other articles. The keywords used, sometimes in combination with each other, are: "Pedagogical Content Knowledge", "Teachers' education", "Programming (Education)", "Student's misconceptions/difficulties in learning to program", "Secondary education", "Why to teach programming", and "How to teach Programming".

The choice of the articles has been mostly dictated by the search of papers published in research journals and presenting research results, as practice in science education research suggest. Also, papers from conference proceedings are considered, where relatively young subjects like informatics find the fastest way to share their results. Despite the call for people who are active in CSER to rely on scientific papers preferably not published in conference proceedings (Randolph, 2007; Lister, 2007), we found that still most of the up-to-date literature is shared through conference papers and therefore the need to rely on them.

The goal of this literature study is to sketch the PCK of Programming, not of specific topics (e.g., variables, functions, etc.), but referring to programming as a subject. As instrument we use the framework introduced earlier, which consists of the four questions (see Fig. 1).

5. PCK of Programming

As previously stated, the PCK of a subject is the knowledge that enables researchers and teachers to better understand the issues related to the teaching and learning of the subject, and consequently provide a better teaching. In this section we give preliminary answers to the core questions uncovering the PCK of programming, using the method described above.

Why Teach Programming?

What are the reasons to teach programming at high school level for non-major students? In a way, this question could also be reformulated as "why should students learn to program at all?". The answer, however, is interesting from a teacher's perspective. If this literature study would focus on students rather than teachers, than the question should be rephrased as "why should I, as student, learn to program?". This question, if properly answered, would also help teachers to motivate students to enrol in informatics courses in a first place. However, this is not the goal of this paper.

Soloway (1993) answers this question by reporting "respected folks' opinions" such as those by Seymour Papert and Alan Kay, arguing that in learning to program one learns powerful problem-solving/design/thinking strategies. This is because when students program, they first need to find a solution to a problem, and then they need to reflect on how to communicate their solution to the machine, using syntax and grammar, through an exact way of thinking (Papert, 1980; Szlávi and Zsakó, 2006). The latter contributes to the students' natural language skills, because they are required to learn to tell, in an unambiguously way, what they want the computer – an unintelligent machine – to perform (Hromokovič, 2006).

Programming involves the ability to generate a solution to a problem. Generating solutions means that one of the learning outcomes is the ability to solve problems and also, if the problem is a big problem, the ability to split the problem into sub problems

and create a generalizable central solution. In addition, the student achieves the ability to create usable, readable and attractive solutions.

Problem solving skills can be deployed to solve "realistic" problems in various domains together with the computing goals (Sims-Knight and Upchurch, 1993; Dagienė, 2005). Transferability of these and other skills is the argument that brought Feurzeig and his colleagues (Feurzeig *et al.*, 1970) to introduce programming as a way to help students to understand mathematics concepts such as: rigorous thinking, variable, function, decomposition, debugging and generalization. Syslo and Kwiatkowska (2006) went further by exploring those mathematics concepts that can benefice from programming, but which have not be included in the (Polish) secondary school curriculum yet. Besides transferability of skills, when learning to program students also acquires a sense of mastery over a technological instrument and establishes contact with some of the deepest ideas of different disciplines such as: science, mathematics and the art of intellectual model building (Papert, 1980). Moreover, as we anticipated earlier on, programming is a new generation subject, which brings together pieces from different areas such as: linguistics, mathematics and economics (Mulder, 2002). This completeness gives students the opportunity to be faced with a multi-disciplinary subject that connects different aspects in a single class. Students could experience the opportunity to delve deeper into previously acquired knowledge, as for example Resnick and Ocko's students (1990) did with the physics concept of friction.

What should Be Taught?

By answering this question we aim to understand what are the core concepts of programming students need to learn. Decisions about what it is needed to teach are usually taken by curriculum and examinations designers. In informatics efforts to define a suitable curriculum have been made since the late '60s (Atchison *et al.*, 1968). However we should consider the different curricular representations (Van den Akker and Voogt, 1994) including: the ideal curriculum, which refers the original ideas and intentions of the designers; the formal curriculum, denoting the written curriculum (documents, materials); the perceived curriculum, indicating the interpretation of the users, especially the teachers, of the curriculum; the operational curriculum, identifying the actual instruction process in the classroom; and the experiential curriculum, which represents students' reactions and outcomes. In this literature study we combine topics suggested from the ideal, the formal and the perceived curriculum (e.g., Gal-Ezer and Harel, 1999; Tucker *et al.*, 2003; UNESCO, 2002; Tucker, 2010), because we think that these together form a more complete and realistic view of what happens in a class.

Rephrasing Romeike (2008), the core of programming is all about problem solving and creating a program as solution. In programming we can distinguish two kinds of knowledge, namely the program generation and the program comprehension (Van Merriënboer and Krammer, 1987; Robins *et al.*, 2003; Mannila, 2007). In the first case, the programmer analyzes the problem, produces and algorithmic solution, and then translates this algorithm into a program code. This means that students should be coached in the

process of problem solving, reflection on this process, and in the development of algorithmic ways of thinking (Feurzeig *et al.*, 1970; Resnick and Ocko, 1990; Sims-Knight and Upchurch, 1993; Dagienė, 2005; Breed *et al.*, 2005; Hromkovič, 2006; Futschek, 2006; Ginat, 2006). As for program comprehension, the programmer is asked to give a demonstration of her/his understanding of how a given program works. We consider then the teaching in secondary school of program generation and program comprehension very important.

Programs are a set of instructions that computers execute in order to perform a task and are written in a programming language. Usually curriculum designers leave the choice of the programming language to teachers, and among secondary teachers there seems to be heterogeneity in the choice of programming languages/paradigms. In the process of learning to program, Govender (2006) identifies, from a technical point of view, three main aspects students need to learn: data, instructions and syntax. Data refers to the concepts of variables and data types for procedural programming, and objects involving attributes and actions for OO programming. As for instructions, the needed understanding is about control structures and subroutines for the procedural programming, and interacting objects and methods in the case of OO programming. Syntax denotes the group of rules that determine what is allowed and what is not within a programming language. Syntax rules determine what it is called the vocabulary of the language, how programs can be constructed using techniques such as loops, branches and subroutines.

Govender's classification, however, does not take care of the modularity and abstraction aspects of programming, as for example Abelson and Sussman (1996) do. They identify three main aspects: primitive expressions, representing the simplest entities that a language is concerned with; means of combination, by which compound elements are built from simpler ones; and means of abstraction, by which compound elements can be named and manipulated as units. These three aspects deal with two kinds of elements: data and instructions. By using these three mechanisms in combination with each other it is possible to formulate complex programs, starting from simpler ones.

A final aspect, equally important, is the semantic of a program, also referred to as the meaning of a program. A semantically correct program is a program that performs the required task. Programs written with different syntax can perform the same semantic task.

What Are the Tearning Difficulties?

In this section we deal with students' different needs and difficulties. Because of the complexity of individuals, different students will have different needs and difficulties. For this reason some of the studies presented might result contradictory, but in fact they present the different realities of different students.

It has been stated several times that programming is a difficult task to achieve (Van Diepen, 2005; Govender, 2006) and often novice programmers hold misunderstanding and misconceptions. In early stages of the learning process a correct program often results as an unexpected surprise (DuBoulay, 1989). By answering this question we aim to

understand the most common problems students have while learning to program. From this knowledge we can attempt to find solutions to prevent or guide these problems. A brief exploration of the most common problems is given.

DuBoulay (1989, p. 283–284) identifies five kinds of difficulties/areas which have a certain degree of overlap in programming learning/teaching, concerning aspects such as motivation and technical aspects. Students' difficulties are: 1) orientation, finding out what programming is useful for and what the benefits to learn to program are; 2) the notional machine (understanding the general properties of the machine that one is learning to control) and realizing how the behaviour of the physical machine relates to the notional machine; 3) notation, which includes the problems of aspects of the various formal languages such as syntax and semantics; 4) structures, understanding the schemas or plans that can be used to reach small-scale goals (e.g., using a loop); 5) mastering the pragmatics of programming (learning the skill to specify, develop, test and debug a program using the available tools).

From a relationship student-computer point of view, Pea (1986) identifies the existence of persistent conceptual language-independent "bugs" in how novices program and understand programs. The starting point of the analysis of conceptual "bugs" is that students have a tendency to *converse* with a computer as if it was a *human* (considered also as the superbug), with consequences such as expecting the computer to interpret students' conversations. Pea distinguishes three different kind of conceptual "bugs": the *parallelism* bug refers to the assumption that different lines in a program can be active or somehow known by the computer at the same time, or in parallel. Another bug is the *intentionality*, for which students believe that computers "go beyond the information given" in the lines of programming code being executed when the program is run. The last bug, *egocentrism*, refers to students' assumption that there is more of their meaning for what they want to achieve in the program than is actually present in the code they have written (e.g., "Don't print what I say, print what I mean!"). Students' conceptions do not guide their attention to consider these problems as relevant reasons for their programs not to work as planned.

Another problem students could face is the paradigm shift (Kölling, 1999a, 1999b; Mazaitis, 1993) in cases where their teacher proposes them to learn more than one programming language with different paradigms (e.g., procedural and object oriented), although this is not advisable in an introductory course at secondary school level. Students usually encounter problems in passing from one paradigm to another, especially from the procedural to the object oriented (but not the vice versa).

Regarding the acquisition of problem solving skills, several papers explore the different difficulties students encounter while trying to generate a solution for a given problem. Novices (Ginat, 2006) tend to maintain local, limited-insight points of view of the problem, leading often to undesirable, erroneous outcomes. It seems that novices fail to realize the importance of a global point of view. Also Weigend (2006) observed how, even when finding a mental or practical solution to a problem, students fail to write a correct program that does the job. The reason might be that students are not trained to translate mental intuitions in a communicative way, or might be connected with the semantic of a program.

Semantic is also considered to be a problematic aspect of programming. This is because it requires the student to put together different parts of a program (variables, expressions, statements, control structure, objects and methods) into a working solution. Semantic is closely related to the debugging activity and the related correctness of a program (Pea and Kurland, 1983), a concept introduced by Dijkstra (1968, 1972). When teachers choose a programming language offering a more complex syntax, students will be faced with both semantic and syntax difficulties (Mannila *et al.*, 2006).

How should the Topic be Taught?

By answering this question we aim to understand what the best approaches to introduce students into the learning of programming are, not only to prevent the above mentioned difficulties/misconceptions, but also to hook students' motivation in an effective and engaging way. As in the previous section, because of the complexity of individuals, different students will have different needs and difficulties. For this reason some of the studies presented might report contradictory results, but actually they propose different teaching approaches to meet different students' learning needs. We cannot conclude which one method is the best, but only highlight those methods which are considered best in different circumstances. This is also directly connected with Shulman's definition, which considers PCK as an armamentarium/repertoire of representations.

Hromovič (2006) suggests that programming is seen as a skill to communicate, in an unambiguously way, a set of instructions to an unintelligent computer. If this process could take place by means of a relatively simple programming language (e.g., Python) offering a simpler syntax than other commonly used programming languages, students could focus more on the semantic aspect of the program and produce fewer syntax errors (Mannila *et. al.*, 2006). Another way to start this learning process could be the use of practical examples, such as rewriting recipes for cooking for a cooking machine (Hromovič, 2006). The process should lead students to write at first simple programs, and then combine the simple solutions together to obtain solution to more complicated problems (Abelson and Sussman, 1996). This approach has the twofold purpose to let the student not only experience the historical development, but also learn the concept of modularity and reusability. Writing a set of instructions to solve a problem is the definition of algorithm. In other words, writing code for a correct mental solution. To achieve algorithmic thinking students should solve many problems, which should be chosen independently from any programming language (Futschek, 2006), and should follow some pedagogical principles (Romeike, 2008). In fact, algorithmic thinking can be successfully introduced without the aid of a computer at all (Bell, Witten and Fellows, 1998; Curzon and McOwan, 2008). However, it happens that students fail to translate their correct reasoning into an unambiguous set of instructions for the machine. To overcome this, students could be coached in analysing their intuitions and connecting them to the designated task (Weigend, 2006).

Linn and Dalbey (1989, p. 58–62) suggest an ideal chain for learning computer programming, which gradually goes from program comprehension and ends with program

generation. The chain has three main links: single language features, design skills, and general problem-solving skills. According to Linn and Dalbey (1989) the ideal chain should start with the understanding of the language features, knowledge that can be assessed by asking students to reformulate or change a language feature in a program so that the program does something slightly different. The second link of the chain consists of design skills, which are a group of techniques used to combine language features to form a program. This chain link also includes templates (stereotypical patterns of code that use more than a single feature) and procedural skills (used to combine templates and language features in order to solve new problems, including planning, testing and reformulating). The third link of the chain, problem-solving skills, is useful for learning new formal systems. Problem-solving skills can be assessed by asking students to solve problems using an unfamiliar formal system such as a new programming language. Though this chain of cognitive accomplishment requires an extensive amount of time it forms a good summary of what could be meant by deep learning in introductory programming (Robins *et al.*, 2003).

To provide novices with a framework for understanding, some model or description of the machine should be introduced, where a model should be designed around each group of novices, distinguished either for their age, background or kind of studies (Du Boulay *et al.*, 1989). Students working with such models excelled at solving some kind of problem more than students without the model (Mayer, 1989). An example could be the metaphor of a black box inside the glass box as a way to present computing concepts to novices. The reason is that novices start programming with very little idea of the properties of the notional machine implied by the language they are learning.

The previous approaches mostly deal with the difficulties and misconceptions presented in the previous section. If we look at approaches which aim at teaching programming in an engaging way, we should refer to the family of programming environments and suited programming languages developed with the main goal to introduce students into the programming practice in active and motivating scenarios. These environments have been specially designed to answer the difficulties students usually encounter when learning to program with normal programming languages (Mannila *et al.*, 2006). The list is quite long and the first efforts have been already made in the early ‘70s. Among the most popular we have Logo and its derivates (Feurzeig *et al.*, 1970; Papert, 1980; Resnick and Okco, 1990; etc.), initially designed to teach mathematics, which has the focus to enhance problem solving skills; Scratch (Resnick *et al.*, 2009) which, based on a metaphor of building bricks and offering much of the same functionality as Logo, allows students to create syntactically correct program, and leaves the students to focus on the semantic aspect; and finally the more modern Alice Greenfoot and Gamemaker (relatively Cooper *et al.*, 2003; Kölling and Henriksen, 2005; Overmars, 2005). These learning environments find their basis in Piaget’s model of children’s learning, where students are fostered to build their own intellectual structures, if provided with the right material. It is then the teacher’s task to find suitable support/stimuli/learning material to use with each of these tools. Some of these languages and environments, however, might not include some structures or topics important to the learning of programming (Murnane and McDougall, 2006).

6. Conclusions and Implications

In the previous section we gave the first preliminary answers to the questions that aim to uncover the PCK of programming.

The first question aims to understand what the reasons to teach programming are. Preliminary answers to our first questions are the following: enhancing students' problem solving skills and offering the students a subject, which includes aspects of different disciplines; use of modularity and transferability of the knowledge/skills; and the opportunity to work with a multi-disciplinary subject.

The second question aims to list the concepts/aspects that a programming curriculum should include. Preliminary answers point to the following concepts/aspects: programming knowledge, which refers to the knowledge of the data, instructions and syntax of a programming language, but also primitive expression, means of combination and means of abstraction; programming strategies, which identify the way syntax is used to create programs to solve problems; and programming sustainability, which refers to the ability to create user friendly and attractive program/software that takes care of ethical and privacy issues.

The third question aims to answer issues relative to the various difficulties students encounter while learning to program, such as a general problem of orientation; difficulty to instruct the machine about the solution of a problem; and tendency to converse with a computer as if it was a human Regarding the solution of a problem, students tend to maintain a local, limited point of view, failing to find a suitable solution.

The fourth question addresses these difficulties, by discussing teaching methods such as possible and effective teaching sequences; offering a simple programming language so students can focus on the syntax; choosing several problems to solve, which should be carefully chosen, independently from any programming language, in order to achieve algorithmic thinking; and teaching by means of suited programming languages or programming environments.

In most of the cases these four 'answers' are not connected with each other, because no explicit attempt to uncover the PCK of programming has been done before, neither on higher or secondary education. The task in portraying the PCK of programming will be to find the answers not only from a general point of view (programming in general), but from the perspective of each of the most frequently taught topics, which are at the heart of learning to program (e.g., variables, functions, etc.). An example will be answering the four questions regarding the teaching of problem solving skills. Despite the fact that some of these answers are available for some concepts, most have still to be studied. Therefore we propose a call for research to portray the PCK of the most commonly taught programming topics.

This literature study constitutes the first phase of a PhD project, which is still in progress. In the second phase it will be attempted to uncover the PCK of Programming for secondary education from an international perspective, through the use of research instruments already deployed in other subjects (Loughran *et al.*, 2001). In the third phase an instrument will be developed to assess to what extent an informatics textbook can support

teachers with low PCK. While the fourth phase of the project will consist of the formulation of an approach to assess the PCK of Dutch teachers. The results of this project will be used to improve teacher training for the subject of programming.

References

Abelson, H., Sussman, G.J. (1996). *Structure and Interpretation of Computer Programs*, 2nd edition. Series MIT Electrical Engineering and Computer Science.

Almstrum, V.L., Guzdial, M., Hazzan, O., Petre, M. (2005). Challenges to computer science education research. In: *Proceedings ITiCSE*, St. Louis, 191–192.

An, S., Kulm, G., Wu, Z. (2004). The pedagogical content knowledge of middle school, mathematics teachers in China and the U.S. *Journal of Mathematics Teacher Education*, 7, 145–172.

Atchison, W.F., Conte, S.D., Hamblen, J.W., Hull, T.E., Keenan, T.A., Kehl, W.B. *et al.* (1968). Curriculum 68: Recommendations for academic programs in computer science: a report of the ACM curriculum committee on Computer Science. *Communications of the ACM*, 11, 151–197.

Bell, T., Witten, I.H., Fellows, M. (1998). Computer Science Unplugged. Off-line activities and games for all ages. `http://unplugged.canterbury.ac.nz`

Breed, e.a., Monteith, J.L. de K., Mentz, E. (2005). Effective learning in computer programming: the role of learners' reflective thinking. In: Samways, B. (Ed.), *35 Years of Computers in Education: What Works? Proceedings of IFIP 8th World Conference on Computers in Education – WCCE 2005*, University of Stellenbosch, Western Cape, South Africa.

Carpenter, T.P., Fennema, E., Peterson, P.L., Carey, D.A. (1988). Teachers' pedagogical content knowledge of students' problem solving in elementary arithmetic. *Journal for Research in Mathematics Education*, 19, 385–401.

Cochran, K.F., DeRuiter, J.A., King, R. A. (1993). Pedagogical content knowing: an integrative model for teacher preparation. *Journal of Teacher Education*, 44, 263–272.

Cooper, S., Dann, W., Pausch, R. (2003). Teaching objects-first in introductory computer science. *SIGCSE 2003*.

Curzon, P., McOwan, P. (2008). Engaging with computer science through magic shows. Paper presented at *The 13th Annual Conference on Innovation and Technology in Computer Science Education*.

Dagienė, V. (2005). Teaching information technology in general education: challenges and perspectives. In: R.T. Mittermeir (Ed.), *Proceedings of International Conference on Informatics in Secondary Schools – Evolution and Perspectives*, Klagenfurt, Austria, 53–64.

Dijkstra, E.W. (1968). A constructive approach to the problem of program correctness. *BIT Numerical Mathematics*, 8, 174–186.

Dijkstra, E.W. (1972). Notes on structured programming. In: Dahl, O.-J., Dijkstra, E.W., Hoare, C.A.R. (Eds.), *Structured Programming*. London and New York, Academic Press, 1–82).

Du Boulay, B. (1989). Some difficulties of learning to program. In: Soloway, E., Spohrer, J.C. (Eds.), *Studying the Novice Programmer*, London, Lawrence Erlbaum Associates, 283–299.

Du Boulay, B., O'Shea, T. Monk, J. (1989). The black box inside the glass box : presenting computing concepts to novices. In: Spohrer, C. (Ed.), *Studying the Novice Programmer*. London, Lawrence Erlbaum Associates, 431–446.

Feurzeig, W., Papert, S., Bloom, M., Grant, R., Solomon, C. (1970). Programming-language as a conceptual framework for teaching mathematics. *Newsletter SIGCUE Outlook*, 4(2), 13–17.

Futschek, G. (2006). Algorithmic thinking: the key for understanding computer science. In: Mittermeir, R.T. (Ed.), *ISSEP 2006, LNCS*, 4226, 159–168.

Gal-Ezer, J., Harel, D. (1999). Curriculum and course syllabi for a high-school cs program. *Computer Science Education*, 9, 114–147.

Ginat, D. (2006). On novices' local views of algorithmic characteristics. In: Mittermeir, R.T. (Ed.): *ISSEP 2006, LNCS*, 4226, 127–137.

Govender, I. (2006). *Learning to Program, Learning to Teach Programming: Pre- and In-service Teachers' Experiences of an Object-oriented Language*. University of South Africa.

Grossman, P.L. (1989). A study in contrast: sources of pedagogical content knowledge for secondary English. *Journal of Teacher Education*, 40, 24–31.

Grossman, P.L., Lynn, P. (1990). *The making of a Teacher: Teacher Knowledge and Teacher Education*. New York, Teachers College Press, Columbia University.

Guzdial, M. (2004). Programming environments for novices. In: Fincher, S., Petre, M. (Eds.), *Computer Science Education Research*, Lisse, The Netherlands, Taylor & Francis, 127–153.

Hashweh, M.Z. (2005). Teacher pedagogical constructions: a reconfiguration of pedagogical content knowledge. *Teachers and Teaching*, 11, 273–292.

Holmboe, C., McIver, L., George, C. (2001). Research agenda for computer science education. In: *13th Workshop of the Psychology of Programming Interest Group*, 207–223.

Hromkovič, J. (2006). Contributing to general education by teaching informatics. In: Mittermeir, R.T. (Ed.), *ISSEP 2006, LNCS*, 4226, 25–37.

Kölling, M. (1999a). The problem of teaching object-oriented programming, Part I, Languages. *Journal of Object-Oriented Programming*, 11, 8–15.

Kölling, M. (1999b). The problem of teaching object-oriented programming, Part II, Environments. *Journal of Object-Oriented Programming*, 11, 6–12.

Kölling, M., Henriksen, P. (2005). Game programming in introductory courses with direct state manipulation. *ITiCSE 2005*.

Kurland, D.M., Pea, R.D., Clement, C., Mawby, R. (1989). A study of the development of programming ability and thinking skills in high school students. In: Soloway, E., Spohrer, J.C. (Eds.), *Studying the Novice Programmer*. London, Lawrence Erlbaum Associates, 83–112.

Lapidot, T., Hazzan, O. (2003). Methods of teaching a computer science course for prospective teachers. *Inroads – The SIGCSE Bulletin*, 35(4), 29–34.

Linn, M.C., Dalbey, J. (1989). Cognitive consequences of programming instruction. In: Soloway, E., Spohrer, J.C. (Eds.), *Studying the Novice Programmer*. London, Lawrence Erlbaum Associates, 58–62.

Lister, R. (2007). The randolph thesis: csed research at the crossroads. *Inroads – SIGCSE Bulletin*, 39, 4, 16–18.

Loughran, J., Milroy, P., Berry, A., Gunstone, R., Mulhall, P. (2001). Documenting science teachers' pedagogical content knowledge through PaP-eRs. *Research in Science Education*, 31, 289–307.

Magnusson, S., Krajcik, J., Borko, H. (1999). Nature, sources, and development of pedagogical content knowledge for science teaching. In: Gess-Newsome, J., Lederman, N.G. (Eds.), *Examining Pedagogical Content Knowledge*. Dordrecht, the Netherlands, Kluwer Academic Publishers, 95–132.

Mannila, L., Peltomaki, M., Salakoski, T. (2006). What about a simple language? Analyzing the difficulties in learning to program. *Computer Science Education*, 16, 3, 211–227.

Mannila, L. (2007). Novices' progress in introductory programming courses. *Informatics in Education*, 6, 139–152.

Marks, R. (1990). Pedagogical content knowledge: from a mathematical case to a modified conception. *Journal of Teacher Education*, 41, 3–31.

Mayer, R.E. (1989). The psychology of how novices learn computer programming. In: Soloway, E., Spohrer, J.C. (Eds.), *Studying the Novice Programmer*. London, Lawrence Erlbaum Associates, 129–159.

Mazaitis, D. (1993). The object-oriented paradigm in the undergraduate curriculum: a survey of implementations and issues. *SIGCSE Bulletin*, 25, 58–64.

Morine-Dershimer, G., Kent, T. (1999). The complex nature and sources of teachers' pedagogical knowledge. In: Gess-Newsome, J., Lederman, N.G. (Eds.), *Examining Pedagogical Content Knowledge*. Dordrecht, the Netherlands, Kluwer Academic Publishers, 21–50.

Mulder, F. (2002). van BÈTA – naar DELTA-discipline (in English: Computer Science: from a BÈTA to a DELTA subject). *Informatica, Tinfon*, 11, 48.

Murnane, J., McDougall, A. (2006). Bad computer science in beginners programming courses: "Considered harmful?" – A case study of the Trufts graphical programming language. In: Watson, D., Benzie, D. (Eds.), *Proceedings of IFIP-Conference on Imagining the future for ICT and Education*, Alesund, Norway.

Overmars, M. (2005). Teaching computer science through game design. *IEEE Computer*, 37(4), 81–83.

Papert, S. (1980). *Mindstorms. Children, Computers and Powerful Ideas*. Basic Books, Inc. Publishers, New York.

Pea, R.D., Kurland, D.M. (1983). On the Cognitive Prerequisites of Learning Computer Programming. Technical report No. 18, Bank Street College of Education, New York, NY.

Pea, R.D. (1986). Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research*, 2, 25–36.

Peterson, P.L., Fennema, E., Carpenter, T.P. Loef, M. (1989). Teacher's pedagogical content beliefs in mathematics. *Cognition and Instruction*, 6, 1–40.

Ragonis, N., Hazzan, O., Gal-Ezer, J. (2010). A survey of computer science teacher preparation programs in israel tells us: computer science deserves a designated high school teacher preparation! *SIGCSE'10 Proceedings of the 41st ACM Technical Symposium on Computer Science Education*. Milwaukee, Wisconsin, USA.

Randolph, J.J. (2007). Computer Science Education at the Crossroads: A Methodological Review of Computer Science Education Research: 2000–2005. PhD Dissertation. Utah State University. Logan, Utah.

Rayner, S., Riding, R. (1997). Towards a categorisation of cognitive styles and learning styles. *Educational Psychology*, 17, 5–27.

Resnick, M., Ocko, S. (1990). *LEGO/Logo: Learning Though and about Design*. Epistemology and Learning Group, E & L Memo No. 8, MIT Media Laboratory, Cambridge.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, November 2009.

Rich, Y. (1993). Stability and change in teacher expertise. *Teacher and Teacher Education*, 9, 137–146.

Robins, A., Rountree, J., Rountree, N. (2003). Learning and teaching programming: a review and discussion. *Computer Science Education*, 13, 137–172.

Romeike, R. (2008). What's my challenge? The forgotten part of problem solving in computer science education. In: Mittermeir, R.T. (Ed.), *ISSEP 2008, LNCS*, 5090, 122–133.

Rovegno, I.C. (1992). Learning to teach in a field-based methods course: the development of pedagogical content knowledge. *Teaching and Teacher Education*, 8, 69–82.

Sanders, L.R., Borko, H., Lockard, J.D. (1993). Secondary science teachers' knowledge base when teaching science courses in and out of their area of certification. *Journal of Research in Science Teaching*, 30, 723–736.

Shulman, L.S. (1986). Those who understand: knowledge growth in teaching. *Educational Researcher*, 15, 4–14.

Shulman, L.S. (1987). Knowledge and teaching: foundations of the new reform. *Harvard Educational Review*, 57, 1–22.

Sims-Knight, J.E., Upchurch, R.L. (1993). Teaching software design: A new approach to high school computer science. Paper presented at *The Annual Meeting of the American Educational Research Association*. Atlanta, GA.

Soloway, E. (1993). Should we teach students to program? *ACM Communications*, 36, 21–24.

Soloway, E., Spohrer, J.C. (1989). *Studying the Novice Programmer*. London, Lawrence Erlbaum Associates.

Stephenson, C., Gal-Ezer, J., Haberman, B., Verno, A. (2005). *The New Educational Imperative: Improving High School Computer Science Education* (Rep. No. Final Report of the CSTA Curriculum Improvement Task Force – February 2005).

Syslo, M.M. Kwiatkowska, A.B. (2006). Contribution of informatics education to mathematics education in schools. In: Mittermeir, R.T. (Ed.), *ISSEP 2006, LNCS,* 4226, 209–219.

Szlávi, P. Zsakó, L. (2006). Programming versus application. In: Mittermeir, R.T. (Ed.), *ISSEP 2006, LNCS* 4226, 48–58.

Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C., Verno, A. (2003). *A Model Curriculum for K-12 Computer Science* (Rep. No. Final Report of the ACM K-12 Education Task Forse Computer Science Curriculum Committee).

Tucker, A.B. (2010). K-12 computer science: aspirations, realities and challenges. In: J. Hromkovič, R. Královič, J. Vahrenhold (Eds.): *ISSEP 2010, LNCS*, 5941, 22–34.

Turner-Bisset, R. (1999). The knowledge bases of the expert teacher. *British Educational Research Journal*, 25, 39–55.

UNESCO (2002). *Information and Communication Technology in Education. A Curriculum for Schools and Programme of Tacher Development*, Paris.

Van den Akker, J., Voogt, J. (1994). The use of innovation and practice profiles in the evaluation of curriculum implementation. *Studies in Education Evaluation*, 20, 503–512.

Van Diepen, N. (2005). Elf redenen waarom programmeren zo moeilijk is (in English: Eleven reasons why programming is so difficult). Tinfon, 14, 105–107.
Van Driel, J.H., Verloop, N., de Vos, W. (1998). Developing science teachers' pedagogical content knowledge. *Journal of Research in Science Teaching*, 35, 673–695.
Van Merriënboer, J.J.G., Krammer, H.P.M. (1987). Instructional strategies and tactics for the design of introductory computer programming courses in high school. *Instructional Science*, 16, 251–285.
Weigend, M. (2006). From intuition to programme. Programming versus application. In: Mittermeir, R.T. (Ed.), *ISSEP 2006, LNCS*, 4226, 117–126.
Woollard, J. (2005). The implications of the pedagogic metaphor for teacher education in computing. *Technology, Pedagogy and Education*, 14 (2)2, 189–204.

M. Saeli received her bachelor degree in computer science from the University of Ferrara (Italy) and her master degree in mathematics and science education from the University of Amsterdam (The Netherlands). Currently she is doing her PhD at the Eindhoven School of Education (Eindhoven University of Technology) on the teaching of programming for secondary school.

J. Perrenet participated in various mathematics education research projects and was involved in the development and innovation of higher technological education for many years. Nowadays he is associated with the Eindhoven School of Education, for teacher training in science education and communication, for coaching PhD students, and for research into mathematics and informatics education. He is also associated with the mathematics and computer science programmes of the TU/e for developmental advice and participates in the project Academic Competencies and Quality Assurance that measures the academic profile of programmes at the TU/e and at other technical universities.

W. Jochems received his master degree in educational psychology and methodology from Utrecht University. He did his PhD in technical sciences at Delft University of Technology (TU Delft). In 1989 he became full professor in educational development and educational technology at TU Delft. From 1993 until 1998 he was dean of the Faculty of Humanities at TU Delft. In 1998 he became dean of the Educational Technology Expertise Centre at the Open University of the Netherlands (OUNL) and full professor in educational technology. From 2006 onwards prof. Jochems is dean of the Eindhoven School of Education and full professor in educational innovation at Eindhoven University of Technology.

B. Zwaneveld, professor in mathematics education and informatics education at the Ruud de Moor Centrum of the Open Universiteit of the Netherlands, expertise centre in professional development of teachers. B. Zwaneveld is professor in mathematics education and informatics education at the Ruud de Moor Centrum of the Open Universiteit. His main fields of interest are the training of intending informatics teachers, the teaching of mathematical modelling and the teaching of mathematics in primary education. He started his career as a mathematics teacher in secondary education. Afterwards he was a course developer in the Faculty of Computer Science of the Open Universiteit. He has a PhD in didactics of mathematics.

Programavimo mokymas vidurinėje mokykloje: pedagoginių dalyko žinių perspektyvos

Mara SAELI, Jacob PERRENET, Wim M.G. JOCHEMS, Bert ZWANEVELD

Pedagoginės dalyko žinios apibrėžiamos kaip žinios, kurios leidžia mokytojams pakeisti jų dalyko žinias į kažką prieinamesnio mokiniams. Straipsnio autorių tikslas – surasti preliminarius atsakymus į klausimus, kuriais siekiama atskleisti pedagogines dalyko žinias apie programavimo mokymą. Pagrindiniai keliami klausimai: dėl kokių priežasčių yra mokoma programavimo; su kuriomis sąvokomis reikėtų supažindinti mokinius; kokios yra dažniausiai pasitaikančios klaidos ir sunkumai, su kuriais susiduria mokiniai, besimokantys programuoti; kaip mokyti programuoti. Autoriai, pasitelkdami atsakymus į klausimus, kurie nėra akivaizdžiai susiję, nori rasti priežastis, kodėl mokyti programuoti reikia jau vidurinėje mokykloje.