The Characteristics of Pedagogical Content Knowledge of Teachers Teaching an Introductory Programming Course

by

Huub de Beer

supervisors:

J. Perrenet
M. Saeli

Eindhoven, August 1, 2009

Abstract

This study captures part of the pedagogical content knowledge of programming of teachers teaching a introductory programming course by means of Context Representations (CoRe). The result of this study are two categorised lists of "Big Ideas" of teaching a first programming course and a CoRe about four of these "Big Ideas". Another characteristic of PCK of programming is the focus on practicality. All respondents agreed: one learns to program by programming. The respondents' teaching practices reflect this focus on practicality through examples, exercises, assignments, course materials, and so on. Furthermore, programming as taught in an first programming course in higher education differs from programming taught in secondary education although both introduce the subject programming.

Contents

1 Introduction

Computer science is a relatively new subject in secondary education in the Netherlands. Only since the late 1990s the subject is taught in secondary education. At the same time the *Consortium Omscholing Docenten Informatica* (CODI) retrained 336 experienced teachers to become a computer science teacher (Schmidt, 2007, 18).

Programming is a substantial part of the secondary computer science education curriculum. Although the CODI trained teachers teach programming for over ten years now, almost nothing is known about *how* they teach programming. In other words the pedagogical content knowledge (PCK) of computer science teachers is largely unknown. With this research project I try to determine part of the PCK of programming of Dutch computer science teachers. The main research question is: *What are the characteristics of PCK of teachers teaching an introductory programming course?*

Initially my research plan focused on Dutch *secondary* computer science teachers. Unfortunately most respondents did not have any experience with teaching computer science in secondary education. As a result this research will be broader in scope to include computer science teachers on all levels of education. In the discussion, however (See Section 6), the focus will be again on programming in secondary education.

After the CODI retraining was finished, no new secondary computer science teachers were educated until 2006 when the three universities of technology in the Netherlands started the Master of Computer Science Education. As said before, almost next to nothing is known about the PCK of programming of Dutch secondary computer science teachers. Internationally the situation is not much better. To fill in this hiatus, Mara Saeli started in 2008 a PhD project on PCK in computer science teaching practices (Saeli, 2008). I collaborated with her in this small research project on the Dutch situation. During my research she also acted as one of my supervisors.

Because almost nothing is known about the PCK of programming of Dutch secondary computer science teachers this research is relevant for teachers and teacher educators alike. Computer science teachers will be able to reflect on their own teaching practice and strengthen their own PCK with the results of this research. Teacher educators can use this research to discuss teaching programming with their students on a more objective level than to base these discussions on their own and students' experiences with programming and programming education. The results of this research can be used to improve teacher education and/or change policy about secondary computer science education in the Netherlands.

Furthermore, computer science in secondary education in the Netherlands is set up to be suited for all students, culture and science minded students alike. As a result, secondary computer science has no entry requirements. There is even no requirement to have successfully finished computer science in secondary education to start a computer science related higher education. Both programming in secondary education as introductory programming courses in higher education try to introduce students to the subject of programming. Nonetheless, a difference between the two is to be expected.

Before continuing with the specific research questions, two important con-

cepts are explained first: programming and PCK. Then, after the research questions, the method used to gather and analyse data is explained. Following the analysis the research questions will be answered. Finally the results are discussed.

2 Theory

2.1 Programming

In this research the definition of "programming" as given in the *Examenprogramma informatica havo/vwo* (Exam program for secondary computer science education set by the Dutch government) (Schmidt, 2007, 43–45) is used. In the *Examenprogramma*, the word "programmeren" (programming) is never used explicitly. However, it states that: "the student knows and is able to apply simple data types, programming structures, and programming techniques" (Schmidt, 2007, 44, translated from Dutch by the author) Schmidt interprets this as the student being able to write a simple web application (Schmidt, 2007, 20). Furthermore knowledge of the system development process is also mentioned (Schmidt, 2007, 45).

Although this definition applies to programming education in secondary education and not to introductory programming courses in higher education, this definition is used to make a connection from programming in higher education to programming in secondary education. A first programming course in higher education is either the start of a range of programming related courses or the only course on programming, depending on the discipline. One would expect that programming in the latter case is comparable to programming in secondary education: programming is an important, but secondary subject. Whereas in the former case, programming is a primary subject.

2.2 Pedagogical Content Knowledge

In 1985 Lee Shulman coined the term *pedagogical content knowledge* (PCK) (Turner-Bisset, 1999, 41). He observed a gap between content knowledge and pedagogy and introduced a new category of knowledge, pedagogical content knowledge, to fill that gap. He defined PCK as going "beyond knowledge of subject matter per sé to the dimension of subject matter knowledge *for teaching*." (Shulman, 1986, 9).

He then continues to include "for the most regularly taught topics in one's subject area, the most useful forms of representation of those ideas, the most powerful analogies, illustrations, examples, explanations, and demonstrations.

(. . .)

[PCK] 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 (. . .) teachers need knowledge of the strategies most likely to be fruitful in reorganizing the understanding of learners." (Shulman, 1986, 9–10)

PCK is an intrinsic part of teachers' experience and often tacit in nature (Loughran, Berry, & Mulhall, 2007; Baxter & Lederman, 1999). Teachers often

do not have the means or experience to articulate the ideas, decisions, actions, etc. underlying their teaching practice that constitutes their PCK (Baxter & Lederman, 1999). As a result determining teachers' PCK is difficult.

Furthermore, PCK as a concept is ill-defined (Loughran, Milroy, Berry, & Gunstone, 2001; Loughran, Mulhall, & Berry, 2004; Loughran et al., 2007; Gess-Newsome, 1999). Or, as Gess-Newsome put it, "PCK has fuzzy boundaries" (Gess-Newsome, 1999, 10). Since the introduction of the concept, many a researcher has used the concept. Thereby often changing the meaning of the concept to include extra or different knowledge categories. Turner-Bisset takes it to the extreme and sees PCK as the superset containing all other knowledge categories for teaching a specific subject (Turner-Bisset, 1999).

Nevertheless, the PCK concept is useful and accepted as such (Loughran et al., 2001). Two observations: experienced teachers have more PCK than inexperienced teachers; and an experienced teacher in one area switching to another area will create more easily PCK in the new area than an inexperienced teacher (Sanders, Borko, & Lockard, 1993). The CODI trained secondary computer science teachers fall into this last category.

As said before, determining teachers' PCK is difficult. Over the years different methods of determining PCK have been developed and used. These methods range from questionnaires, concept mapping, interviewing and observation. Most often, however, researchers combine different methods to determine teachers' PCK (Baxter & Lederman, 1999). Although time consuming, these *multi-methods* have the potential to capture the vague concept PCK best.

In this study the original definition of the concept PCK as given by Shulman in 1986 is used. Later alterations, limits or extensions of the concept do not change the underlying idea of the concept: teachers' knowledge about how to teach a subject best. The goal of this study is to capture part of teachers' knowledge how to teach introductory programming best. There is no need to adapt the original definition to reach this goal.

3   Research questions

The main question of this exploratory research is: *What are the characteristics of PCK of teachers teaching an introductory programming course?* This question is separated into three specific research questions.

1. What characteristics are visible in Dutch computer science teachers' PCK of teaching introductory programming courses?

2. How do these characteristics relate to programming in the *Examenprogramma*?

3. How do the characteristics of Dutch computer science teachers' PCK relate to the backgrounds of these teachers?

First the PCK of programming of Dutch computer science teachers teaching introductory programming courses is topic of research. What characteristics are visible? Next, the relation between these characteristics and programming in secondary education is examined: What is the difference between teaching an introductory programming course in higher education and teaching programming in secondary education?

	workshop I		workshop II	
	part A	part B	part A	part B
secondary education	1	1	0	1
hogeschool	2	2	4	3
university	2	1	0	0
total	4*	3*	4	4
	5		6	

*: One of the respondents is a teacher in secondary education, a teacher educator and also a teacher in higher education. Hence the difference between the total number of respondents and the sum of the respondents per educational level.

Table 1: The respondents and their background per (part of the) workshop

Finally the background of the respondents is taken into account. What is the difference between teachers teaching in different levels of education? Does it matter if a teacher teaches computer science and engineering students or students studying programming unrelated subjects?

4   Method

4.1   Respondents

The relevant population for this study is all computer science teachers in the Netherlands. The selective sample taken from this population is all Dutch computer science teachers visiting the NIOC congress[1]. The respondents taken from this sample were participants of two workshops held during the NIOC congress.

The workshops were advertised on http://www.informaticavo.nl, a website targeting secondary computer science teachers (see Figure 3 on page 20). Readers of the invitation were asked to apply beforehand, almost no-one did, however. On the congress itself the flyer with the invitation was also circulated. As the NIOC congress was visited mostly by computer science teachers in higher education, anyone at the congress who wanted to participate in the workshop, was accepted. There was no active selection of respondents from our side.

As a result, almost all respondents teach Computer Science courses in higher education. Most work on a hogeschool (similar to a college) and teach computer science and engineering courses to technology minded students. One respondent works on a hogeschool for Media and ICT, he teaches programming to a group of students with a different, less technology minded, background.

All the respondents teaching in higher education share a similar traditional background in computer science. One respondent in the first workshop originally teaches on a university at the Faculty of Mathematics and Computer Science. Later he became a teacher educator and also started to teach in secondary education. There was only one CODI trained teacher, he took part in the second part of the second workshop. His original did teach music and mathematics in secondary education.

In Table 1 the respondents are listed per workshop per educational level. In the first workshop a total of 5 people participated from both university and hogeschool level. One participant also teaches in secondary education and is a teacher educator.

The second workshop started out with 4 hogeschool teachers. After the break, a CODI trained teacher joined. He did not have any influence on the listing of "Big Ideas" however. In this workshop a total of 6 people participated. Over all 11 people participated in the two workshops.

All respondents were male and from different ages.

4.2   Procedure

The general design of this study was a semi-structured group discussion with two small groups.

4.3   Instruments

The instrument used in this study is the *Context Representation* developed by a group of Australian researchers (Loughran et al., 2001; Mullhall, Berry, & Loughran, 2003; Loughran et al., 2004; Loughran et al., 2007): One or more *content representations* (CoRes) are constructed through structured discussions. In these discussions a small group of teachers is asked to identify the "big ideas" of a certain topic these teachers teach. Then, for every big idea, several standard questions are discussed, for example:

  - "What you intend the students to learn about this idea

  - Difficulties/limitations connected with teaching this idea

  - Teaching procedures (and particular reasons for using these to engage with this idea)

  - Specific ways of ascertaining students understanding or confusion around this idea" (Mullhall et al., 2003, 7–8).

These big ideas, the questions, and the answers are put into a matrix and form the CoRe. Due to time constraints this instrument was slightly adapted. Per structured group discussion only two "Big ideas" were discussed in more detail.

Furthermore, to answer the question how the characteristics of PCK relate to the definition of programming in the *Examenprogramma*, the lists with "Big Ideas" were annotated.

In the *examenprogramma* (Schmidt, 2007, 44) programming in secondary education is defined with three aspects: 1. simple data types, 2. simple programming structures and 3. simple programming techniques. The "Big Ideas" listed during the workshop will be marked with a number $(1, 2, 3)$ to denote the category of the "Big Idea". Some "Big Ideas" do not fit into any category, they are not marked. The respondents did not know of this definition and categorisation.

As this is a definition and categorisation of programming in *secondary computer science education* and the respondents listed "Big Ideas" in a first programming course in *higher education*, some "Big Ideas" do not fit the definition well. The definition speaks of *simple* data type, *simple* programming structures, and *simple* programming techniques, some of the "Big Ideas" listed are beyond simple. Those more advanced "Big Ideas" are also marked with a ⋆. I marked "Big Ideas" as advanced based upon my experience in teaching programming in secondary education.

category	group discussion I		group discussion II	
	#	*(advanced)*	#	*(advanced)*
[1] simple datatypes	9	(2)	3	(1)
[2] simple programming structures	6	(2)	9	(2)
[3] simple programming techniques	12	(9)	10	(5)
[-] none	6	(−)	7	(−)
total	32	(13)	29	(8)

Table 2: Number of listed "Big Ideas" per category and total. The number of advanced ideas is put between parentheses.

Listing and categorizing "Big Ideas" from two different workshops with two different but similar groups of respondents resulted in similar lists both in number of total items as in number of items per category, advanced ideas included. Often after one respondent mentioned one concept, other similar or related concepts were also mentioned. So chance plays a role in listing "Big Ideas". Nevertheless, listing and categorising "Big Ideas" is probably reliable.

Both groups came up with more or less all fundamental concepts found in programming languages as "Big Ideas". As almost all respondents teach programming courses as part of a programming related discipline, the advanced concepts listed are to be expected. Listing and categorising "Big Ideas" is probably valid.

The CoRe-instrument seems also reliable: similar groups of respondents in two different structured group discussions answer the eight standard questions similarly. Questions A, D and H were answered in more detail and resulted in a diverse mix of answers from the different respondents Other questions, most notably B and C, but also E and F, seem less relevant to the respondents. The questions were often answered shortly and there is almost no diversity in the answers given.

The instrument is used successfully to capture PCK of science teachers in Australia(Mullhall et al., 2003) and, also given the broad definition of PCK, it is probably valid for capturing PCK of teachers teaching an introductory programming course. On the other hand, the subject programming differs from the subjects researched using this instrument: programming is a skill whereas the other topics, like chemical reactions, are more theoretical in nature.

5 Results

5.1 Compiling "Big Ideas": Results

In Figure 1 (page 8) the "Big Ideas" gathered during the first structured group discussion are listed and categorised. In Figure 2 (page 9) you will find the "Big Ideas" of the second structured group discussion. These two lists of "Big Ideas" are one result of this study: according to the respondents these are the most important concepts taught in an introductory programming course.

In Table 2 (page 7) these lists of both group discussions are summarised and compared by putting the number of "Big Ideas" per category and the total number of ideas in one table. The number of advanced ideas per category is put between parentheses after the number of ideas per category.

Figure 1: "Big Ideas" listed during the first structured group discussion

1. object orientation[3*]
2. class[1,2*]
3. inheritance[3*]
4. polymorphism[3*]
5. variable[1]
6. constant[1]
7. sequence[2]
8. selection[2]
9. iteration[2]
10. state
11. pre- and postcondition[3]
12. specification[3]
13. algorithm
14. searching strategies[3*]
15. sorting[3*]
16. backtracking[3*]
17. representation of data
18. tree[1*]
19. list[1]
20. type[1]
21. assignment[2]
22. array[1]
23. index[1]
24. pointer[1*]
25. compiling versus interpreting
26. syntax
27. semantics
28. design strategies[3*]
29. modular development[3*]
30. top-down[3*]
31. exception handling[2*]
32. debugging[3]

The "Big Ideas" listed are categorised by:
[1] : simple data structures,
[2] : simple programming structures,
[3] : simple programming techniques,
* : advanced level.
Unmarked ideas do not fit into any of the three categories.

Figure 2: "Big Ideas" listed during the second structured group discussion

1. object orientation[3]*
2. variable[1]
3. constant[1]
4. control structure[2]
5. if, if-then-else[2]
6. repetition[2]
7. sequence[2]
8. condition[2]
9. debugging[3]
10. documenting[3]
11. designing[3]*
12. function[2]
13. stepwise refinement[3]*
14. bottom-up[3]*
15. top-down[3]*

16. trial and error[3]
17. history of programming
18. memory
19. interfaces[2]*
20. syntax
21. algorithm
22. expression[2]
23. event[1]*
24. event-handler[2]*
25. compiling
26. translating
27. execution
28. testing[3]
29. specification[3]

The "Big Ideas" listed are categorised by:
[1] : simple data structures,
[2] : simple programming structures,
[3] : simple programming techniques,
* : advanced level.
Unmarked ideas do not fit into any of the three categories.

In both group discussions, the respondents started their list of "Big Ideas" with object oriented programming in general, a fairly advanced concept. After asking what exactly "object oriented" means the respondents concluded that this concept is not a "Big Idea" in a *first* programming course. During the session the respondents regularly came up with other advanced topics and concepts. It was difficult for them to stay focussed on beginner level programming.

Both groups of respondents came up with similar "Big Ideas": they listed the basic elements of programming languages like variables, constants, iteration, selection, and functions. There is one exceptional difference, in the second group discussion less Big Ideas from the category simple data types were listed than in the first group discussion.

Furthermore, often after the mention of one concept other respondents reacted with other similar and related concepts. Directly after "variable" followed "constant"; "sequence" resulted in "selection" and "iteration"; "control structure" in the second group discussion resulted in "if" and "if-then-else", "repetition", "sequention", "condition" and so on.

Although the advanced concepts listed did vary more between the two group discussions they were not fundamentally different. Some concepts listed, like "history of programming" or "compiling versus interpreting", did not fit into the three categories of the small definition of programming in the *Examenprogramma*. They refer to the broader context of programming, computers and computer science.

5.2   Compiling the CoRe: Results

In Figure 3 (page 16) you will find the CoRe created from the transcript of the first structured group discussion about the "Big Ideas" *array* and *iteration*. The CoRe from the second structured group discussion, about *expression* and *variable* is put in Figure 4 (page 17). Some general remarks can be made about the eight different standard questions:

A *What would you like your students to learn about this "Big Idea"?*

This question asks the respondents to get to the essence of the "Big Idea". There are "complete" answers, capturing the concept in one or two sentences and very specialised answers focussing on a small but important detail of the concept. Different respondents with different students focus on different details. For example in the first group discussion, "Big Idea" array, one respondent wants that his students learn that "it is just a pointer (in the programming language C)", while another respondent wants his students to learn that "it is a representation for real-world problems". Although these answers differ greatly, all respondents did understand and accept these answers given by others.

B *Why is it important for your students to learn about this "Big Idea"?*

As most "Big Ideas" in a first programming course are fundamental concepts of programming languages, there is really no question why these "Big Ideas" are important, as one of the respondents put it, "without it they are unable to program".

For these basic "Big Ideas" this question is thus not relevant. On the other hand, for the "Big Ideas" listed without annotation and some of the advanced

concepts, this question is more relevant. Because then teachers make a choice to teach that concept (or not).

C *What do you know more about this "Big Idea" (and your students do not need to know yet)?*

By discussing the "Big Ideas" variable and array the respondents of the group discussions agreed that they know about implementation details of these concepts their students did not have to know. Besides implementation, the respondents also mentioned some advanced concepts like complex types and the formal syntax of expressions.

D *Problems/difficulties relative to the learning of this "Big Idea"*

As with the first question, the respondents gave many different answers. The "Big Idea" as a whole is never the problem, but there are a lot of problems students have encountered when learning the concept. As a result, the answers are specific smaller problems associated with learning the "Big Idea" under question.

E *Knowledge about students' thinking that influences your teaching of this "Big Idea"*

This question was difficult to answer. The respondents did agree, however, that the students' background does matter. It is important to connect with the students and supply a context and problems where this "Big Idea" does make sense to this group of students.

F *Other factors that influence your teaching of this "Big Idea"*

This question was only answered in the second structured group discussion. There were three factors mentioned: practical aspects of school like colleagues, standard course material, etc.; professional experience, for example with a coding standard; and most concepts are interwoven with each other. This last factor was mentioned more often during the discussions: all these basic concepts belong together and one concept can not be seen apart from the others. It was difficult for the respondents to separate the one concept under question from the whole.

G *Teaching procedures (and particular reasons for using these to engage with this "Big Idea")*

This question was only answered in the second group discussion. The respondents agreed: practicality is key, both in problems, examples and assignments as in the way students learn, that is, by doing.

H *Specific ways to remove students misunderstanding or confusion around this "Big Idea"*

Different respondents gave totally different answers. However most answers have in common that students should experience programming, for example using tools like IDEs[2], compilers and something called "live programming". The answers were not targeted specifically on the "Big Idea" under question but more general in scope.

5.3   The characteristics of PCK of teachers teaching a first programming course

What are the characteristics of PCK of teachers teaching a introductory programming course? First, the important concepts taught in a first programming course are important characteristics. In Figures 1 and 2 you will find the "Big Ideas" listed by the respondent during the two structured group discussions. These "Big Ideas" include fundamental programming language concepts, advanced programming and software-engineering concepts and some concepts related to the context of programming like the history of programming languages.

Other characteristics are the constructed CoRes with the results of the structured group discussions about array and iteration in the first group discussion and expression and variable in the second discussion. Given this CoRe, another characteristic of PCK of teachers teaching an introductory programming course becomes visible: the focus on practicality. The learning-by-doing mentality is shared among all teachers in the two structured group discussions. Programming is a practical skill gained through experience, one learns to program by programming. However, the teachers go a step further than that: they use practical exercises, examples and teaching practices suited for their students. The background of the students does matter.

Practicality does not mean professionalism, however. At best, the "Big Ideas" listed only touch the realm of professional software development. In a first programming course a student makes his or her first steps in the trade of programming.

All teachers agree that they themselves should know implementation details and formal notation of expressions but this knowledge is not suited for their students in a first programming course. The students should learn the basics of programming first and students are protected in course materials, examples and problems from the gory details of programming. That does not mean students have an easy time programming: "Big Ideas" like testing, debugging, trial-and-error all refer to learning by solving mistakes. Some teachers even give their students faulty examples and problems to force the students to make mistakes.

5.4   Relation to programming in the *Examenprogramma*

What is the relation of the characteristics of PCK of teachers teaching an introductory programming course and programming in the Examenprogramma*?* From both lists with "Big Ideas" it became clear that teaching "object oriented programming" is a hot topic among computer science teachers in higher education. In both structured group discussions the respondents started with object oriented programming. After focussing on a first programming course, however, the respondents started summing up fundamental programming language concepts.

Both groups did have difficulties to stay focused on beginner level programming, times and again more advanced topics were listed. Clearly a first programming course in higher education is more advanced than anything expected in secondary education given the definition of programming in the *Examenprogramma*.

Furthermore the teachers listed 21 "Big Ideas" (61 total) that did not fit in the definition of programming in the *Examenprogramma*. According to the

teachers there is more to programming that simple data types, simple programming structures and simple programming techniques.

Although there is a difference between a introductory programming course and programming in secondary education, there is no indication that this difference influences the characteristics of the PCK of programming, except for the advanced and extra topics taught. The two teachers teaching in secondary education also stressed the need for practicality, they also mention protecting their pupils from gory details, etc.

5.5 Relation of the characteristics to the background of the teachers

What is the relation of the characteristics of PCK of teachers teaching an introductory programming course and the background of the teachers? As most respondents share a similar background in computer science and work in higher education, there is no visible difference between the PCK of these teachers. In this study there is a difference, however, between teachers teaching students studying computer science (and engineering) and students studying business information systems on the one hand and the teacher teaching students studying Media and ICT on the other hand. The latter uses other examples, assignments and so on than the others: the background of the students and the study does matter for the characteristics of PCK of a teacher.

The same goes for the one computer science teacher in secondary education: his input differed from the input of the others. He gave slightly different answers to the eight standard questions. However this difference could be superfluous as there obviously is a difference between teaching programming to pupils in secondary education and students in higher education. The two different contexts (secondary education and higher education) result in different perspectives on teaching programming and hence on the articulation of elements of PCK. That does not mean that the PCK of the secondary computer science teacher is fundamentally different from that of the teachers in higher education.

6 Discussion

The main research question of this research is: *What are the characteristics of PCK of teachers teaching an introductory programming course?* First of all, the lists with "Big Ideas" and the CoRe constructed from the structured group discussions about array, iteration, expression, and variable are all characteristics of PCK of the respondents teaching an introductory programming course.

Secondly, even when discussing "Big Ideas" of a *first* programming course, programming and teaching programming in higher education is of a more advanced level than programming and teaching programming in secondary education. In other words there is a gap between programming on a secondary education level and first year higher education.

But is that problematic? Successful completion of computer science in secondary education is not an entry requirement to study a computer science related discipline. It seems that an introductory programming course in higher education teaches more than one learns in secondary education. What then is the value of teaching programming in secondary education? Just a general introduction into the subject of programming without practical application or use in

13

higher education? This may be an indication for re-evaluation of programming in secondary education into a more important subject.

Another characteristic is a focus on practicality in teaching programming. The teachers agree that practical examples, exercises, visualisations, course materials, etc. help students to learn to program. Students learn by programming, by trial and error, and the problems should fit the students background to maximize the effectiveness of the experience of solving the problem.

Furthermore there is more to programming than as defined in the *Examenprogramma*. Thirty percent of the "Big Ideas" listed does not fit the, clearly, limited definition of programming used in secondary education. It is uncertain if the definition of programming also does not fit the experience of teaching programming of secondary computer science teachers.

Because the respondents almost all teach in higher education, not much can be said about computer science teachers in secondary education. Nevertheless, the method of capturing PCK used in this study seems to capture (parts of) respondents' PCK of teaching programming. Nevertheless, some of the standard questions of the structured group discussion were not that relevant. It seems advisable to adapt the instrument for use in the context of teaching (introductory) programming. To capture the PCK of programming of secondary computer science teachers a similar set up can be used: use the CoRe instrument with different groups of secondary education computer science teachers.

The respondents all stressed that it was refreshing to think about the different "Big Ideas" in isolation instead as part of a whole programming language. It will probably be as refreshing for secondary education teachers. Furthermore, if this research will be done with secondary education teachers, the results of both researches can be compared.

Although the instruments used seem to work to capture PCK of the four discussed "Big Ideas", the captures PCK is far from a complete picture of PCK of programming. To create a more complete picture, more research is needed.

What does this research mean for secondary education computer science? The connection with higher education seem to be far from ideal. Also the definition of programming in the *Examenprogramma* appears to be too narrow. It would be advisable to rethink the implementation of the subject programming in the second education computer science curriculum.

Finally the instrument used in this study to capture PCK is also a useful tool in teacher education: the respondents, teachers, were forced to think about a topic they know by heart, something they do not regularly. Furthermore it is a good way to communicate different perspectives and opinions on teaching programming as long as the group in the structured group discussions is diverse in nature.

Notes

[1] Nationale Informatica Onderwijs Congres, see for more information: http://www.nioc.nl/
[2] Integrated Development Environment (IDE): software to support professional software development. An IDE often contains an editor, project management software, compilers and interpreters, etc.

References

Baxter, J. A., Lederman, N. G. (1999). Assessment and measurement of pedagogical content knowledge. In J. Gess-Newsome N. Lederman (Eds.), *PCK and science education* (pp. 147–161). Kluwer Academic Publishers.

Gess-Newsome, J. (1999). Pedagogical content knowledge: an introduction and orientation. In J. Gess-Newsome N. Lederman (Eds.), *PCK and science education* (pp. 3–17). Kluwer Academic Publishers.

Loughran, J., Berry, A., Mulhall, P. (2007). Pedagogical content knowledge: what does it mean to science teachers? In R. Pintó D. Couso (Eds.), *Contributions from science education research* (pp. 93–105). Springer.

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

Loughran, J., Mulhall, P., Berry, A. (2004). In search of pedagogical content knowledge in science: Developing ways of articulating and documenting professional practice. *Journal of Research in Science Teaching, 41*(4), 370–391.

Mullhall, P., Berry, A., Loughran, J. (2003, December). Frameworks for representing science teachers' pedagogical content knowledge. *Asia-Pacific Forum on Science Learning and Teaching, 4*(2), 1–25.

Saeli, M. (2008). *Research proposal esoe – mara saeli.* Eindhoven School of Education (ESoE).

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*(7), 723–736.

Schmidt, V. (2007a). *Handreiking schoolexamen informatica havo/vwo (assistance school exam computer science for havo and vwo).* http://www.slo.nl/downloads/Handreiking_Informatica_DEFINITIEF.pdf/, SLO, Enschede.

Schmidt, V. (2007b). *Vakdossier 2007 informatica (file on computer science in secondary education 2007).* http://www.slo.nl/themas/00108/341330003_Vakdossier_2007_informatica.pdf/, SLO, Enschede.

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

Turner-Bisset, R. (1999). The knowledge bases of the expert teacher. *British Educational Research Journal, 25*(1), 39–55.

A CoRe of structured group discussion I: Array and Iteration

Table 3: The CoRe of the first structured group discussion: Array and Iteration

		Big Idea I: Array	Big Idea II: Iteration
A	*What would you like your students to learn about this Big Idea?*	• An array is a whole series of variables in one compact object • It has an upper bound and a lower bound • It is just a pointer (in C) • It is a list, a sequence of variables with some specific characteristics • It has an index to access elements • It is a representation for real-world problems	• "What arrays are for data, iteration is for control" • Arrays and iteration are connected • The operational semantics of an iteration • The looping variable and its scope • There are different variants of iteration statements: for, while, repeat, foreach, etc.
B	*Why is it important for your students to learn about this Big Idea?*	• You need arrays to write non-trivial programs • It is elementary and it is connected to other (basic) concepts like an index • "It is one of the simplest things to make things more difficult." • "I am unsure if my students really need to know the concept"	
C	*What do you know more about this Big Idea (and your students do not need to know yet)?*	• Memory allocation and memory management	
D	*Problems/difficulties relative to the learning of this Big Idea*	• Indexing and related problems like the upper bound and the lower bound • Does the array stars at 0 or at 1 • Constant length or variable length arrays • Multi-dimensional arrays • Copying of arrays • When using an array as a parameter to a function, is the array passed to the function by reference or by value • An index itself can be a meaningful value (as in a histogram)	• Breaking out of a loop with break, continue, exit • Misleading keywords like while and do: what do they mean? • When the looping condition becomes false half way during the execution of the loop, the loop does not stop then. • Nested loops • Students put a semicolon at the end of the for-definition and on the next line they write the content of the for-loop. • Input via a loop: forgetting to read the user input again in the loop.
E	*Knowledge about students' thinking that influences your teaching of this Big Idea*	• Provide a context in which arrays are useful and needed • students avoid using arrays	
F	*Other factors that influence your teaching of this Big Idea*		
G	*Teaching procedures (and particular reasons for using these to engage with this Big Idea*		

		Big Idea I: Array	Big Idea II: Iteration
H	*Specific ways to remove students misunderstanding or confusion around this Big Idea*	• Provide a context in which arrays are useful and needed: students then understand why they need arrays • I introduce arrays by looking at a bunch of variables. I connect to the concept of variable to explain the array	

B CoRe of structured group discussion II: Expression and Variable

Table 4: The CoRe of the second structured group discussion: Expressions and Variable

		Big Idea III: Expression	Big Idea IV: Variable
A	*What would you like your students to learn about this Big Idea?*	• What an expression is, how it works and some examples, nothing more than that. • How to create an expression in some programming language: every language it is a little different; the syntax of expressions. • An expression is a combination of variables, operators and other expressions. An expression is an expression of expressions.	• The concept of a variable, not just the word "variable", but what it is. • A variable is a memory location • Every variable has a type • What is the scope of a variable • The assignment of a value *to* a variable • Every variable has a (chosen) name
B	*Why is it important for your students to learn about this Big Idea?*	• It is a basic method to express that your program has to do something • Expressions are used in a selection as a condition • It is the most important part of a programming language	• Without it, one is unable to program
C	*What do you know more about this Big Idea (and your students do not need to know yet)?*	• The formal syntax of expressions, it is an abstraction level they do not understand. • Bitwise operators • I almost never use the term "expression" in my classes although there is an expression on almost every line in a program.	• Memory allocation • References and pointers, they do not need to know then in the beginning • Complex types (like records and objects), sometimes students try to solve problems using only simple types while they better use a complex type, but they do not know them yet.

Continued on next page

		Big Idea III: Expression	**Big Idea IV: Variable**
D	*Problems/difficulties relative to the learning of this Big Idea*	• Calling a method after they looked up the definition of the method in the documentation. Some students copy the whole declaration instead of using just the name. • Combining expressions to more complex expressions, the hierarchy of operators and when to use parentheses • Assignment operator versus comparison operator: they are used to write $A = B$ as in mathematics, but that is the assignment operator. To compare two expressions, you have to use $A == B$ instead. • Knowing the exact syntax of all the operators, like division with the percent sign.	• Using the same name over and over • It is an abstract concept • The scope of a variable is a fundamental problem • It is difficult to choose meaningful names for variables
E	*Knowledge about students' thinking that influences your teaching of this Big Idea*	• The background of the students is important: there is for example a difference between computer science engineering students and business information systems. We adapt our courses and course materials to better connect with our students.	
F	*Other factors that influence your teaching of this Big Idea*	• There are standard course materials I have to use • My colleagues, we decide together what we teach • "Expressions" is not a separate subject: it is interwoven with other concepts during the course	• Experience with coding standards: how to name variables consistently and with (extra) meaning like Hungarian notation
G	*Teaching procedures (and particular reasons for using these to engage with this Big Idea*	• The more practical the assignments and examples the better. • Teaching by example, practical examples like programming a robot with one or two sensors. The conditions then are straightforward. • Trial and error • Visualisation, for example Boolean expressions with lights and switches	

		Big Idea III: Expression	Big Idea IV: Variable
H	*Specific ways to remove students misunderstanding or confusion around this Big Idea*	"Live programming": writing simple programs live in front of the class and all students follow me on their own computers. They make errors just copying my program and their neighbours have different errors. They then start working together to solve these errors. - Programming errors not caught by tools like editors and compilers, those are the worst especially when the program does work. - Give students a program full of errors to solve and trace the errors back given the error log of the compiler.	- A box with Lego to visualise memory and variables - Referring to machine schemas as used in school mathematics - Live programming - Refactoring with an IDE like Eclipse to visualise the scope of variables - Using good examples and making errors

C Invitation Workshop

Figure 3: Invitation for the Workshop published on http://www.informaticavo.nl, a website for secondary computer science teachers in the Netherlands. Furthermore, the invitation was also circulated at the NIOC. This flyer invites secondary computer science teachers to participate in our workshop and share their experience and ideas about teaching programming with us and more importantly with each other.