Educ Inf Technol (2010) 15:87–107
DOI 10.1007/s10639-009-9098-8

Levels of thinking in computer science: Development in bachelor students' conceptualization of algorithm

Jacob C. Perrenet

Published online: 2 June 2009

Abstract How do we know if our students are beginning to think like computer scientists? In a first study we defined four levels of abstraction in computer science students' thinking about the concept of algorithm. We constructed a list of questions about algorithms to measure the answering level as an indication for the thinking level. This list was presented to various groups of Bachelor computer science students. The mean answering level increased between successive year groups as well as within year groups during the year, mainly from the second to the third level. Student-level estimations provided by teachers fell in the same range as the level measurements, but level growth was not detected in their estimations; level estimation appeared very difficult for lecturers. The reliability of the instrument proved to be satisfactory. To investigate the validity, a follow-up study was done with a small heterogeneous group of Bachelor students. They answered the same questions and were successively interviewed to check whether they understood the terms they used. Their understanding proved to be satisfactory, sustaining the validity of the instrument. In the first study little relation was found between thinking levels and regular test results on algorithm-oriented courses. Supposedly, besides levels on the dimension of abstraction, levels on concretizing, analyzing and synthesizing are also relevant. A broader framework for future research is being proposed.

Keywords Computer science education · Abstraction · Level of thinking

1 Introduction

Computer science can be characterized as a *science of abstraction* (Aho and Ullman 1992). *Abstraction* has been recognized as a fundamental and essential principle in

J. C. Perrenet (✉)
Eindhoven School of Education, Eindhoven University of Technology, P.O. Box 513,
Eindhoven 5600 MB, The Netherlands
e-mail: j.c.perrenet@tue.nl

computer science problem-solving and software development (Haberman 2004). The key for a successful career in computing therefore lies in the ability to perform abstract thinking and to exhibit abstraction skills (Kramer 2007). Would it be possible to follow students in their development along the dimension of abstraction capabilities and distinguish abstraction levels in their thinking about core concepts? Various computer science concepts can be studied in an educational context and several researchers have made a start. To give a few examples, Hammond and Rogers (2007) investigated the concept of the computer at the transition from primary to secondary education, Papastergiou (2005) studied the concept of the Internet in secondary education, Aharoni (2000) looked at the concept of data structure in the context of higher education and Hazzan (2002) did research into the concept of computability, also in higher education. Our focus in this study, like Haberman's (2004), is the concept of *algorithm* in higher education.

We will start this article with a closer look at what we mean by abstraction in general and abstraction in computer science in particular. Next we will describe our study of students' levels of thinking about the concept of algorithm; see also Perrenet et al. (2005) and Perrenet and Kaasenbrood (2006). The main research question is the following: *Is it possible to distinguish levels of abstraction in students' thinking about the concept of algorithm and to do so in a reliable and valid way?* We constructed an instrument for measuring these levels: a questionnaire consisting of seven semi-open questions together with a method for scoring the answering levels. After securing its reliability, it was used to measure the mean level of several year groups of Bachelor students at several moments and test predictions about level development. As a didactically-relevant side-step, we will describe the outcomes of estimations of the students' answers to the questionnaire, provided by their teachers. The question here is to what extent do they have insight into their students' thinking levels? To check the validity of the instrument we did a follow-up study: a small representative group of students answered the questions of the questionnaire while thinking aloud, after which we interviewed them about their answers to investigate whether they really understood the concepts they had used. In the discussion of the results of both studies, we will look for didactical consequences and reflect on the sufficiency of the dimension of abstraction for the explanation of the results. This reflection will lead to the introduction of a broader framework for future research.

2 Abstraction in computer science education

Several definitions and interpretations are used in literature for abstraction and levels of abstraction. Even within the context of the same discipline it is possible to have more than one prevailing interpretation. For instance in mathematics education (Tall and Thomas 2002) abstraction level has three interpretations: (1) the quality of the relationships between the object of thought and the thinking person; (2) the degree of complexity of the concept of thought; (3) the reflection of the 'process-object duality', where operations on objects and relations between objects become objects themselves on a higher level. According to Kramer two aspects of general definitions of abstraction are relevant in computer science. On the one hand the act or process of leaving one or more properties of a complex object out of consideration so as to attend to other properties. On the other hand the process of formulating general concepts by abstracting common properties of instances (Kramer 2007). According to Dale and Walker two kinds of abstraction are utilized in computer science: (a) procedural abstraction, which involves the separation of the logical properties of an action from the implementation details, and (b) data abstraction, which involves the separation of the logical properties of data from the implementation details (Dale and Walker 1996). Extrapolating from these two kinds, 'the separation of logical properties of an object or process from the implementation details' would make a good candidate for a computer science definition of abstraction. However, we prefer the definition used in another educational research project at the Eindhoven University of Technology. In the ACQA (Academic Competences and Quality Assurance) project the following definition worked quite satisfactorily: *Abstracting is the bringing to a higher aggregation level of a viewpoint (statement, model, theory) whereby it can be made applicable to more cases. The higher the aggregation level, the more abstract the viewpoint.* The definition was accepted by the staff from a variety of engineering disciplines, including computer science. It was used for characterizing the levels of abstraction of thinking and acting in a series of disciplines (Meijers et al. 2005).

Within the context of education it is important to know that, when a student has reached a certain level of thinking, the lower levels still exist and are incorporated; lower levels can be evoked if necessary. Questions or problems do not necessarily evoke the highest available level of thinking. Therefore it is better to speak about the level of thinking in relation to a certain problem. Depending on the problem, a student with a high level of thinking has the possibility but not the obligation to react on the highest possible level (see e.g. Hazzan 2002).

Narrowing it down to the concept of algorithm, we propose four levels of abstraction in students' thinking. On the one hand that is more than the two levels distinguished by Haberman et al. (2005). They argue that an algorithm can be viewed as an operational process entity (embodying a "how" view), as well as an object entity that embodies an I/O relationship (and a "what" view). On the other hand it is less than the seven levels needed by computer science experts to characterize the discipline as a whole (ACQA Project Group 2007); see Appendix. Our abstraction levels are:

1. *Execution* level: the algorithm is a specific run on a concrete specific machine; its time consumption is determined by the machine.
2. *Program* level: the algorithm is a process, described by a specific executable programming language; time consumption depends on the input.
3. *Object* level: the algorithm is not connected with a specific programming language; it can be viewed as an object (versus process); while constructing an algorithm the data structure and the invariance properties are used; meta properties such as termination and 'patterns' (algorithmic modules) are relevant; time consumption is considered in terms of magnitude of order as function of the input.
4. *Problem* level: the algorithm can be viewed as a black box; the perspective of thought is 'given a problem, which type of algorithm is suitable?'; problems can be categorized to suitable algorithms; a problem has an intrinsic complexity.

3 Measuring levels of thinking about the algorithm concept

In this section we will describe the educational context: a selection of algorithm-oriented courses of the Bachelor curriculum of computer science at the Eindhoven University of Technology. Next, the development of the proposed levels of abstraction is described, as well as the construction of the questionnaire along with its scoring system. We will conclude with the description of our test subjects and formulate three hypotheses about the expected results.

3.1 Educational context

Five algorithm-oriented Bachelor courses were selected (Table 1), in which students were supposed to develop their understanding of the concept of algorithm, its construction and its analysis.

3.2 Levels of abstraction

Originally we proposed three abstraction levels for the algorithm concept, the *program* level, the *object* level and the *problem* level. These abstraction levels were discussed with the lecturers of the five courses involved in the study. An extra, lower, level was suggested: the *execution* level. This resulted in the four levels mentioned above.

The lecturers were the most outspoken about the difference between levels 2 and 3. They also expected these two levels of understanding to be the most common among the Bachelor student group. This opinion was shared with the majority of other teachers involved with the subjects of Table 1: faculty instructors for giving help with solving problems and student assistants for assisting with instruction and the correction of assignments. They gave their opinions during short interviews. Some teachers made extra abstraction level distinctions, such as the difference between knowing an algorithm for a certain task and being able to prove that a

Table 1 Algorithm-oriented courses

Subject	Study phase	Contents
Program Realization 1	Trimester 1.1	Embedding elementary algorithms into larger programs and encoding into a specific programming language.
Design of Algorithms 1	Trimester 1.2	Systematical construction of algorithms using the Guarded Command Language.[a]
Design of Algorithms 2	Trimester 2.1	Programming from a formal specification with stepwise refinement to the use of simple standard algorithms.
Design of Algorithms 3	Trimester 2.3	Mathematically analyzing algorithmic run-time. Using advanced techniques of algorithmic design and a number of standard algorithms.
Complexity	Trimester 3.1	Understanding algorithm complexity as well as problem complexity. Transforming problems within complexity classes.

[a] See Kaldewaij (1990)

certain algorithm can complete a certain task in a certain order of time. Or such as thinking about a sequential machine performing an algorithm or thinking about a parallel one. One lecturer's remark was that the ability to use pseudo code and axiomatic notation required a certain level of precision, rather than a certain level of abstraction. The discussions with the lecturers inspired us in reflecting on the contents and the format of the items of the questionnaire.

3.3 Construction of the questionnaire

The original questionnaire consisted of 11 items. The starting item is worded as follows:

0. Give your definition of 'algorithm'.

The other ten items—we call them proposition items—have another format. There is a general introduction: *Mark whether you agree or disagree with the following proposition and give a supporting argument. So, only 'agree' or 'disagree' is not sufficient. If necessary the option 'both are possible' can be chosen, provided that an argument is given. Only choose 'I don't know' if you cannot answer the question because of lack of knowledge.*

All ten items are followed by the four alternatives *Agree*, *Disagree*, *Agree and disagree are possible, I don't know,* and room for the required supporting argumentation. It is mainly the supporting argumentation that is used for further analysis rather than the choice of the answer itself. In this way the method differs from standard use of multiple-choice questions.

This questionnaire was presented to the three Bachelor year groups in the first trimester just after concluding their subject test on Program Realization 1, Design of Algorithms 2 or Complexity (Table 1). These subject tests were the regular preliminary examinations, three hour sessions with theoretical and practical assignments on the subject of the course. A random sample of 5% of the first trimester questionnaire output was scored by two raters. The inter-rater agreement at item level appeared to be too low, so differences and doubts were discussed and scoring rules were refined. This first sample was not used in further analysis. Another 5% sample still resulted in unsatisfactory agreement. However, after removal of four out of the ten proposition items, the degree of agreement between the two raters, measured by Cohen's Kappa, was .64. Also on the remaining list—item 0 and six proposition items—a third rater scored satisfactorily consistently with the other two (.64 and .70). Cohen's Kappa is a measure that corrects the influence of chance. A minimum of .60 is considered acceptable.

We will present the final list of six proposition items. Together with the item 0 (Give your definition of 'algorithm'), this list was used as the questionnaire in further data collection and analysis.

1. An algorithm is a program, written in a programming language.
2. Two different programs written in the same programming language can be implementations of the same algorithm.
3. The correctness of an algorithm can generally be proven by testing the implementation on cleverly selected test cases.
4. A suitable quantity to measure the time needed for a certain algorithm to solve a certain problem is the time needed in milliseconds.

5. The complexity of a problem is independent of the choice of the algorithm to solve it.
6. For every problem it is possible that, in the future, algorithms are discovered which are more efficient by an order of magnitude than the algorithms known up to now.

3.4 Construction of the scoring system

The items were constructed in such a way that it was expected that arguments would be given at various levels. However we did not succeed in constructing a list of items with the possibility of argumentation on all four levels for every item. For items 0 to 4 a level score from 1 to 3 is possible, for items 5 and 6 a score from 1 to 4. In Table 2, as an example, the refined scoring rules are given for three items. The students' thinking level is calculated from the argumentations of the list as a whole. As there is no information about the distances between the levels, we calculated a student's thinking abstraction-level as the *median* of the series of item level scores. If no level was detectable for more than half of the item series (missing or unclear), the data were considered to be insufficient to calculate a students' level.

3.5 Measurement design

Table 3 shows which students took the test at which points of time. All three year groups were measured at the end of their first trimester. The first year group was measured for the second time at the end of the second trimester and the second year group was measured for the second time at the end of their third trimester. Again, these measurements took place at the end of the regular subject test sessions (preliminary examinations). In total 398 questionnaires were gathered (out of 461 students who completed the relevant courses).

3.6 Teachers' estimations of students' thinking levels

Three teacher categories were involved in the teaching of the algorithm courses: lecturers (one per course), faculty instructors (instructors for short) for giving help with solving problems (none to four per course) and student assistants for assistance with instruction and correction of assignments (none to eight per course). Sometimes teachers were involved in various courses as well as in various roles. We asked them to fill in the questionnaire two times, from the perspective of the average successful student and from the perspective of the average unsuccessful student. From the argumentations we could calculate the supposed thinking level as we did with the student data, two times per subject this time.

3.7 Predictions about level development

Our main hypotheses are:

1. *The abstraction level of successive year groups will increase.*
2. *The individual abstraction level will increase during a year's programme.*

Table 2 Examples of detailed answer level scoring

Item	Answer characteristics	Answer level
0	A process with only one input, and only on one machine	1
	A program in a specific language, which can be run with all possible input	2
	A series of steps (abstraction from programming language)	3
	Unclear or missing	x
1	+ An algorithm equals an execution equals a program, on a machine or on a virtual machine	1
	+ Implementation or effectuation in a programming language; program equals algorithm; executable on a machine or by hand	2
	-Implementation into differing languages	3
	± Implementation or effectuation etc., is clearly mentioned (see above)	2
	± Implementation into differing languages	3
	?, +, -, ± Without an argument; unclear or missing	x
6	+ Computers will become faster	1
	+ By optimization of an actual program or by use of a better programming language	2
	+ By different architectures for computers / by optimization of a computational method (not related to a specific programming language); the solving algorithm determines the complexity, the algorithm is placed above the problem	3
	−One specific counter-example is given	3
	−The black-box approach emerges / upper bounds and lower bounds as well as problems are placed above problems; a problem possesses an intrinsic complexity	4
	± The arguments for disagreement should emerge clearly; a problem possesses an intrinsic complexity	4
	? (Without further explanation)	x
	? (The word ‘complexity’ is not understood)	x
	? (The word ‘complexity’ is understood, but the ‘complexity of a problem’ is not)	3
	?, +, −, ± (Without an argument, unclear or missing)	x

Key to symbols used above:

+ = Agree, − = Disagree, ± = Agree and disagree, ? = I don’t know; x = No answer level detectable

3. *The individual grade on a subject test will correlate to the individual abstraction level.*

For the first hypothesis the measurement results of the three groups in the first trimester will be compared (Table 3, column 2). For the second hypothesis the two consecutive measurement results of the first year group and the results of the second year group will be compared (Table 3, row 2 and 3). For the third hypothesis the individual measurement results will be compared with the algorithm subject scores from the same moment.

We will also explore the extent to which the students’ teachers have insight into the development of their students’ thinking levels. Will their estimations fall within the same range as the actual measurements? Will their estimations follow the same trends as the actual measurements (between year groups and within year groups)?

Table 3 Measurement design

Year group	Trimester 1	Trimester 2	Trimester 3
1	Algorithm course 1 Q.11	Algorithm course 2 Q.7	
2	Algorithm course 3 Q.11		Algorithm course 4 Q.7
3	Algorithm course 5 Q.11		

The algorithm courses 1 to 5 refer to the courses in Table 1 chronologically

Q.11 and Q.7 refer to the long, first version of the questionnaire and the shorter, second one, Q.7 being part of Q.11

Are their level estimations for high achieving students higher than for low achieving students?

4 Results of level measurements

Generally, it took the students about 15 min to answer the seven questions. From the answers the level could be calculated for almost 85% of the students who filled in the questionnaire; the other students were too often unclear in their argumentation for the proposition items or gave no argumentation at all.

4.1 Level development

The first hypothesis was that *the abstraction level of successive year groups would increase*. Table 4 shows the percentages of thinking level scores for the year groups 1, 2 and 3 at the end of the first trimester.

Indeed, in higher years, the thinking level generally is higher: Spearmann's Rank Correlation Coefficient Rho=0.48, significant at 0.01 (two-tailed). Almost every student has a median answer level of 2 to 3; levels under 1.5 or above 3 are non-existent.

Our second hypothesis was that *the abstraction level of the same group would increase during the year*. Table 5 shows the general thinking level growth of the first year group and the second year group. The percentages are given of students with a higher level, the same level or a lower level at the second measuring moment, compared to the first.

Table 4 Abstraction level of successive year groups

Year group in trimester	Student percentage with level score				Number of students
Level score	1.5	2	2.5	3	
Year group 1 in 1.1	4	50	6	40	67
Year group 2 in 2.1		21	7	72	58
Year group 3 in 3.1		8	2	90	72

Table 5 Abstraction level growth during the year

Year group (trimesters)	% with increased level	% with the same level	% with decreased level	Number of students
1 (1.1 to 1.2)	48	44	8	36
2 (2.1 to 2.3)	34	56	10	48

Clearly most students reach a higher level or stay at the same level: according to the Wilcoxon Signed Ranks Test significant in both cases at .05 (two-tailed, with $Z=-2.47$ and $Z=-2.27$ successively).

4.2 Relation between individual level and grade

Our third hypothesis was that *the individual grades on the various subject tests would correlate with the abstraction level*. Table 6 shows the correlations between grade and level for the five subject tests. Only for one subject does a small but significant (at 0.05) Spearmann's Rank Correlation appear: for Complexity. For the other subjects, correlations are non-significant and close to 0 (Design of Algorithms 1, 2, 3 and Program Realization 1).

4.3 Teachers' estimations of students' thinking levels

In total 19 teachers were involved in the algorithm courses and 13 of them filled in the questionnaire (3 student-assistants, 6 instructors and 4 lecturers). For the lecturers, guessing students' answers appeared to be a difficult task. In four out of five cases, the guessed answers for the unsuccessful students were missing or the level was not detectable. They explained that they really did not know how *unsuccessful* students would answer. Moreover they proclaimed that they gave their own answers for the *successful* students' case. The instructors and the student-assistants did not have these problems.

Table 7 summarizes the guessed-answer levels. When we look at these data from the perspective of the three hypotheses about the students' results, few relations emerge. The guessed-answer abstraction level of the successive year groups does *not* increase (compare 1.1 with 2.1 and 3.1 in the second and third column). Also, the

Table 6 Correlation between thinking level and test grade

Subject test	Rank correlation	Number of students
Program Realization 1	.14	58
Design of Algorithms 1	.05	63
Design of Algorithms 2	−.05	44
Design of Algorithms 3	.09	72
Complexity	.27*	72

* = significant at 0.05 (two-tailed)

Table 7 Teachers' guessed-answer levels

Trimester	Average guessed-answer level	
	For unsuccessful student	For successful student
1.1	2	3
1.2	2	2.75
2.1	2	3
2.3	2	2.75
3.1	2	3

guessed-answer abstraction level of year groups does *not* increase during the year (compare 1.1 with 1.2 and compare 2.1 with 2.3 in the second and third column). However, on the other hand, the level for the successful-student answer guesses is clearly higher than for the unsuccessful-student ones (compare the second and third column). Also, the guessed levels fall in the same range of 2 to 3 as the actual thinking levels as measured (compare with Table 4).

5 Follow-up study with interviewing a small group

The main goal of this follow-up study was to get qualitative support for the *validity* of the results from the large group. We wanted to investigate to what extent the students really understood the computer science terms they had used in their written answers. In the first study we did an analysis on a small amount of data from many students from three Bachelor year groups; in this follow-up we did qualitative (deep) analysis on a large amount of data from only a few students from the three year groups. Nine bachelor students (paid volunteers) were found willing to participate; three were at the end of their first year, two at the end of their second year and four at the end of their third year. High achieving students as well as moderate ones were included. These students were asked to complete the questionnaire, while thinking aloud, and subsequently were interviewed about their answers. The reason for adding the thinking aloud task on top of the writing task was to give extra information to the interviewer, for use later on. According to Van Someren et al. (1994) thinking aloud does not disturb the thinking process for short, not too complex tasks, so there was no risk that this extra task would influence the thinking level.

5.1 Tasks and procedure

In Table 8 the various activities are given with an approximate timetable. The mean time planned for a session was 50 min.

The first study had given insight into what kind of answers generally could be expected, so specific interview questions could be prepared and adapted to the students' responses to investigate the students understanding of the specific terms

Table 8 Interview scheme with time planned

Activity	Time
Introduction	1 min
Training in thinking aloud	4 min
Part I: Completion of questionnaire (thinking aloud)	20 min
Explanation of follow-up procedure	2 min
Part II: Interview about answers on items 0 to 6	7×3 min
Closing	2 min
	Total time 50 min

used. As an example we will give some typical answers with prepared follow-up questions for item 3.

The wording of item 3 is as follows: *The correctness of an algorithm can generally be proved by testing the implementation on cleverly selected test cases.* A common answer with explanation was: 'Agree, exhaustive testing gives you a good idea about correctness.' In this case the following questions were prepared:

- Exhaustive testing, how is that to be done?
- Which tests do you do?
- How do you draw a conclusion from different test results?
- What do you mean with 'a *correct* program'?
- How do you know that, with the implementation, no properties that could influence the test results were added?
- How do you know that you tested all possibilities?

Other common answers with explanations are: 'Disagree; exhaustive testing is often difficult; you should prove it.' And: 'Disagree: you should prove it by using the pre/post condition technique.' Prepared questions were:

- Why is exhaustive testing often difficult?
- What should be proved according to you?
- Why would proving it lead to a better result; what do you accomplish by proving it?
- Do you have a guarantee that a correct algorithm's implementation will function correctly?

5.2 Analysis of interviews

The sessions were taped and typed out. The abstraction level of the answers was scored, with use of the written answers (as in the first study). The interview results were analyzed to determine for every student which computer science terms were used and whether the used terms were understood. For the level of understanding the following categories were used:

- The student understands the term well or reasonably well.
- The student does not understand the term.

Table 9 Terms used and degree of understanding of the terms

Terms Students:	F1	F2	F3	S1	S2	T1	T2	T3	T4
algorithm	+	+	+	+	+	+	+	+	+
implementation	+	+	+	+	+	+	+	+	+
program. language	+	+	+	+	+	+	+	+	+
different programs	+	+	+	+	+	+	+	+	+
correctness	+	±	+	+	+	+	+	±	+
proving	+	±	+	+	+	+	+	±	+
proving technique	+	+	+	+	±	±	+	+	+
probl. complexity	±	±	±	−	±	+	±	−	+
algor. complexity	±	+	±	±	+	+	+	+	+
order of magnitude	+	±	±	±	+	+	+	+	+
GCL			±	±		±			+
pre/post condition			+					+	+
NP				+		−		±	
specification			±		+				
abstract						+			+
quantum computer				−					−
syntax							−	+	
design					+				
steps		+							
definition		+							
concept									+
formal language				+					
precise						+			
compiler				±					
natural language								+	
semantics									+
ϕ-scheme[a]				−					
state						+			
assembly instruct.				+					
upper bound		±							
greedy					±				
dynamic programming					±				
lower bound								+	
P								+	

Key to symbols used:

−The student does *not* understand the term;

± Some doubt remains as to whether the student understands the term or not;

+ The student understands the term well or reasonably well;

Empty cell: The student does not use the term;

Fi = first year student; Si = second year student; Ti = third year student

[a] ϕ-scheme = an algorithm design method in GCL by use of invariants

– It is not totally clear whether the student understands the term or not.
– The student does not use the term.

Scoring and analyzing were done per item and in changing, random order to counteract possible context effects (the influence of an evaluation of part of a student's work on the evaluation of another part).

5.3 Students' understanding of terms used

In Table 9 the terms, as used by the students, are listed with a score for whether the student does understand the term or not.

The most important result is that, generally, the students appear to understand the terms they use. In only about 5% of the cases is it clear that their understanding is insufficient; in about 75% of the cases their understanding of the term is evaluated as (reasonably) good. In 20% it does not become clear within the given time. Most lack of clarity about the students' understanding is found with the term 'complexity'.

Individual variation While some terms are used by all the students—mainly because the terms are part of the item propositions—there are many terms that are used by only some of the students or even by only one. This result points to the variation of individual cognitive structures and processes. Considerable variation also became visible, even between students from the *same* year group, when they were interviewed about the same topic. For instance, asked for examples of problems that cannot be solved more efficiently, we observed[1]:

F1- 'The computation of the product of two numbers, when it is only allowed to use adding and subtracting, that cannot be done more efficiently than we know now.'
F2- 'Very simple problems do exist, such as just printing something on the screen that cannot be done more efficiently.'
S1- 'When you have to go along an unsorted list, then you have to look at all the elements and you cannot do that more efficiently.'
S2- 'When asked for a program that gives as a result the answer 3. I would write: result becomes 3. Faster is not possible.'
T1- 'The travelling salesman problem it is proved to be an NP-problem.'
T2- 'Looking up something in a telephone book.—A computer cannot do better than binary search.'

As a last example of variation we conclude with parts of the interview results of the two students S1 and T4 who bring up the potential power of quantum computers, although without knowing much about it (see Table 9). While S1 more or less puts this aspect aside, T4 includes it in his 'philosophy'. They are interviewed about their reactions to item 6 with proposition *For every problem it is possible that, in the future, algorithms are discovered which are more efficient by an order of magnitude than the algorithms known up to now.*

[1] Fi = first year student; Si = second year student; Ti = third year student.

S1: ‘I disagree, because it cannot always be done more efficiently than known now.’

Interviewer: ‘Why do you think so?’

S1: ‘For some problems you can show that it cannot be done more efficiently. When you have to go along an unsorted list, than you have to look at all the elements and you cannot do that more efficiently. Therefore it is not possible to find something more efficient for it. Yes, there are such things like quantum computers, but I do not know exactly how those things work and I doubt if that really is relevant.’

T4: ‘I agree, because you cannot make assumptions about the future and especially not about computing devices and computing methods think of quantum algebra.’

Interviewer: ‘What is quantum algebra?’

T4: ‘Apparently they have invented a nice algebra for quantum computers and it appears that you can look through a sorted list in less than log (N) time. In short, you can even look up something in an unsorted list the number of necessary comparisons between the elements of the list and the element you are looking for would be even smaller than the number of elements in the list. Thus you don’t have to treat them all. Very odd. I have not got the slightest idea of how it works, but it appears to be possible. Maybe they will come up with something new again instead of quantum algebra, maybe there is something else God knows how fast you are able to solve things then. Never say “never”!’

6 Conclusion and discussion

Our main results are the following:

- We succeeded in the construction of an instrument for the measurement of students’ thinking level (mean answering abstraction level) for the concept of algorithm; the instrument consists of a series of items and a scoring system.
- The levels detected fell within the range from program level to object level (level 2 to 3).
- The reliability of the instrument proved to be satisfactory: the scores given by several raters correlated well enough.
- The validity of the instrument proved to be satisfactory: generally the students going through the measurement procedure did use the relevant computer science terms with understanding.
- Within a students’ year group the level generally increased during the year.
- For successive year groups the level was generally higher.
- Level estimations done by teachers fell within the same range as the level measurements, but level growth was not detected in their estimations; level estimation was very difficult for lecturers.
- Little relation was found between subject test results and thinking levels.

We will discuss the conclusions in more detail.

6.1 The usefulness of the instrument

Looking back upon the measurement, we think the measurement output of 85% is acceptable.

Kramer articulates the need for a set of abstraction tests aimed at checking student progress, checking teaching techniques and potentially as an aid for student admission selection (Kramer 2007). In our opinion our instrument is not yet ready to fulfil this need; however the *approach* has proved to be fruitful. As a first step, extension with items about other core concepts and aimed at detection of more levels is necessary. Of course, interviewing students as in our study should validate new items. Otherwise one would risk measuring only memorized standard definitions and standard procedures instead of real understanding. One would risk measuring only instrumental understanding and not relational understanding as it is called in mathematics education (Skemp 1976).

In future development a solution should be sought for the 'jargon' problem. This concerns the wording of the propositions. Two of our items contained the term 'complexity' (see item 5 and 6). In our analysis it was not always clear whether a student used the term in the computer-science sense or in the ordinary sense. In these cases we decided the student argumentation was not scorable. In hindsight, it would have been better to use an alternative wording without jargon, if possible. A candidate term would be 'laboriousness'. By using such a term the analysis of more student responses might have been possible.

6.2 Students' thinking levels and level growth

Only levels 2 and 3 were detected: the program level and the object level. This is in accordance with the results of Haberman et al. (2005). An interesting question for further research is the following one: Are levels 2 and 3 really the most common levels or is level 1 (the execution level) prominent for pre-university students and is level 4 (the problem level) prominent for experts or already for students at Master level? On the one hand, part of the explanation for not detecting level 4 is in the characteristics of the items and the method of measurement. For only two of the seven items was an answer at the fourth level possible, which is of influence for the possible range of the mean answering level (actually the median) that was calculated. For the emergence of this highest level as the mean, more items should have been constructed with the possibility of answering at that level. Another possibility would have been to use another operationalization, for instance defining the thinking level of a student as the highest answering level that is reached at least for two items. On the other hand, the lowest level did not show, although this answering level was possible for every item. The students involved had followed one semester or more of computer science education. It would be interesting to look further for the existence of the lower execution level at the actual start of the computer science program or at the end of pre-university education. In that case, however, the questionnaire would need some adaptation: the freshman computer science vocabulary is limited.

Within a student year group, the level increased on average during the year, although a large number of students also remained at the same thinking level. Measuring the same groups after a longer period could reveal clearer results. One could argue that filling in the same questionnaire twice could create a learning effect, explaining the level growth result as an artefact. This would certainly be true when feedback was given; however this was not the case. On top of that, they filled in the questionnaire directly after conclusion of their subject test. Because filling in the questionnaire was supposedly of marginal value for the students compared to the value of the subject test itself, we are sure of the reality of the results. At all events, more frequent measurement of the same group of students would require the construction of parallel items.

Successive year groups showed a higher level. A year group was defined as the group taking a test belonging to the curriculum year. However, slow students in their second year sometimes have to take a first year test again. And fast students in their second year sometimes already take a third year test. More detailed data gathering and data analysis, taking these circumstances into account, could reveal clearer results.

6.3 Teachers' level-estimations

The results of the teachers filling in the questionnaires, from the successful and from the unsuccessful student perspective, were meagre. Neither level growth for successive year groups, nor level growth within year groups, was visible. Rater variability could be a disturbing factor. As some teachers took part in several relevant courses, this could have been diminished by involving only these teachers and involving them more than once. In any case, level differences *could* be measured and within the same range as the actual students' levels. Guessing the students' answers appeared to be difficult. The lack of relation between thinking level and subject test success in the student data partly explains this phenomenon. The task proved to be harder for the lecturer group than for the instructors and student assistant group. A typical difference between these two groups in the teaching process is the distance from the student group and the student work. A lecturer generally has less interaction with the individual students and their work. This points to the importance of interaction in the teaching process in order to understand the students' thinking processes; teaching should not be a one-way communication. As Van Hiele pointed out for mathematics education, the danger always exists that a teacher speaking to a class uses terms that have another meaning for the listeners, because of their differing levels of understanding. Without interaction the teacher will be unaware of the misunderstanding (Van Hiele 1986). More research is needed into this phenomenon within computer science education.

At this point we would also like to take a stand against the use of multiple-choice examinations without investigations of the meaning of their responses. Our validity study showed us, although we were not looking for this explicitly, how much variation exists in students' thinking processes. Students of the same year group showed very different ways of thinking about the same concepts. They used different concepts and different kinds of argumentation leading to the same answers or they even used comparable argumentation to come to different answers. The multiple-choice test format does not uncover all these thought processes, unless the items are tested on students in interview sessions. An alternative is that the examiner asks for an argumentation for the choice of alternative each item, as happened in our measurement procedure.

6.4 Relation between levels and grades

Generally we found little relation between course grade and level of abstraction of thinking at the time (see Table 6). We will look for an explanation. Perhaps it is *precision* more than *abstraction* that is required for algorithmic thinking as one of the lecturers remarked? We will take a broader perspective.

Computer science is more than abstraction alone Besides *abstraction*, solving computer science problems generally requires *analysis* (like analyzing a problem situation), *synthesis* (like designing or synthesizing a solution) and *concretization* (like realizing or concretizing a solution into a specific programming language). This also applies to the problems students have to handle in algorithmic-oriented courses. Analysis, synthesis and concretization are as important as abstraction. In the ACQA project (mentioned in the introduction) these four types of activity—analyzing, synthesizing, abstracting and concretizing—were used as characteristics of the academic way of thinking and acting, in computer science as well in other scientific disciplines, with the following definitions (Meijers et al. 2005):

- Analyzing is the unravelling of phenomena, systems or problems into sub-phenomena, sub-systems or sub-problems with a certain intention. The greater the number of elements involved, or the less clear it is what the elements of the resulting analysis are, the more complex the analysis.
- Synthesizing is the combining of elements into a coherent structure which serves a certain purpose. The result can be an artefact, but also a theory, interpretation or model. The greater the number of elements involved, or the more closely-knit the resulting structure, the more complex the synthesis.
- Abstracting is the bringing to a higher aggregation level of a viewpoint (statement, model, theory) through which it can be made applicable to more cases. The higher the aggregation level, the more abstract the viewpoint.
- Concretizing is the application of a general viewpoint to a case or situation at hand. The more aspects of a situation that are involved, the more concrete the viewpoint.[2]

We suppose that the students' levels in all these four dimensions are relevant for their grades and that, together, they would be related to these grades much more than abstraction alone. So, while Kramers (2007) calls for the annual measurement of students' abstraction abilities, we think a broader endeavour would be even more interesting and more relevant: measurement of students' level of thinking in the *four* dimensions of analyzing, synthesizing, abstracting and concretizing. In the next section we will give an impression of the results of the ACQA project which could be used.

[2] Note that *analyzing* is not treated as the opposite of *synthesizing*, and also that *concretizing* is considered to be an activity independent of *abstracting*.

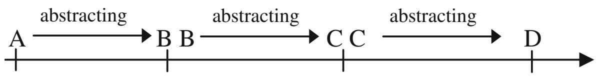

Fig. 1 Construction principle for the ladder of abstraction

Levels in four dimensions In the ACQA project, scales were constructed to measure the level of student activities asked for in the various curricula. With these scales, lecturers were interviewed about their ambitions concerning their courses. In our opinion these scales, or parts of them, could be used to describe students' levels of thinking and to develop an expanded questionnaire inspired by these. We will conclude with the description of the construction principle for levels along the four dimensions resulting in 'ladders' and add the four ladders for computer science to the appendix. While the construction *principle* is generic over all disciplines, the resulting ladder is discipline-specific: it ranges from the lowest level of academic acting and thinking to the highest level in a specific discipline. The four ladders are constructed in relation to central concepts in the discipline. Each ladder is constructed with recurrent steps, where the result of a previous step becomes the object of the activity in the subsequent step. For the activity of abstraction, the ladder looks as in Fig. 1, where A is the example, on the basis of which the scale is constructed, and B, C, and D are reformulations of the example on a higher level of abstraction.

In the Appendix we can see that ACQA's abstraction ladder, although with a greater number of levels than ours, is closely related to our scale of four levels. Actually, one of the results of the ACQA project can account for the only small but significant relation between course grades and thinking levels: at the third year course about complexity (Table 6). It appeared that, compared to the other programming courses, the course about complexity treated the greatest number of abstraction levels within the same course (ACQA Project Group 2007). So, the dimension of abstraction was more important for this course than for the other programming courses.

In conclusion, theories of students' abstraction levels of thinking and the construction of instruments to measure these levels can give insight into students' development in the study of computer science. However, in our opinion, *abstraction* is not the only dimension that is relevant. In computer science, as in all scientific disciplines, c*oncretizing, synthesizing, and analyzing* are just as important and should therefore be included.

Acknowledgements This work is based on earlier work:

Perrenet, J.C., J.F. Groote & E. Kaasenbrood (2005). *Exploring Students' Understanding of the Concept of Algorithm: Levels of Abstraction;* In: Proceedings of the 10th annual SIGCSE-conference on Innovation and technology in computer science education, 64–68; Caparica, Portugal. © ACM 1-59593-024-8/05/0006. http://acm.org/10.1145/1070000/1067467.

Perrenet, J.C. & E. Kaasenbrood (2006). *Levels of Abstraction in Students' Understanding of the Concept of Algorithm: the Qualitative Perspective*; In: Proceedings of the 11th annual SIGCSE-conference on Innovation and technology in computer science education, 270–275; Bologna, Italy. © ACM 1-59593-055-8/06/0006. http://acm.org/10.1145/1150000/1140196.

We thank our colleagues Bert Zwaneveld and Nico van Diepen for their helpful comments at an earlier version of this article.

Appendix: Four dimension with ladders[3] from the ACQA project

Dimension abstract

Abstracting is the bringing to a higher aggregation level of a viewpoint (statement, model, theory) whereby it can be made applicable to more cases. The higher the aggregation level, the more abstract the viewpoint.

1.1 Sorting program running on a given machine at a given time

1.2 Running time of the program

2.1 Sorting program running on a given machine under idealized circumstances

2.2 Pure execution time in clock seconds

3.1 Sorting program, interpreted as a series of elementary calculation steps in a programming language, running on an idealized machine

3.2 Elementary calculations spent in terms of the programming language

4.1 Sorting program when regarded as a sequence of elementary calculations in a mathematical language, running on a Turing machine

4.2 Elementary calculation steps spent

5.1 Sorting program, regarded as a sequence of elementary calculations in a mathematical language (running on a Turing machine) with all possible inputs

5.2 Order of magnitude of the number of required calculation steps as a function of the order of magnitude of the size of the input (i.e., the order of complexity of the sorting algorithm)

6.1 All possible algorithms that solve sorting problems 6.2 Complexity order of the sorting problem

Dimension concrete

Concretizing is the application of a general viewpoint to a case or situation at hand. The more aspects of a situation that are involved, the more concrete the viewpoint.

1.1 An optical effect (‘depth of field’) to be realized in a computer game

1.2 Description of the effect in terms of image processing (‘blurring’)

2.1 An image processing effect

2.2 Description of the effect in terms of mathematical operation(s) on pixels

3.1 Mathematical processing on pixels

3.2 Algorithm for the *efficient* realization of this process (e.g., separable convolution)

4.1 Algorithm for executing the mathematical process

4.2 Implementation of the algorithm in a graphical pipeline model, taking scheduling aspects into account (memory access bandwidth)

5.1 Algorithm for the execution of a mathematical operation in a pipeline model

[3] The generic construction principle is explained in section 6.4.

5.2 Coding the pipeline model for a specific pixel shader

Dimension analytic

Analyzing is the unravelling of phenomena, systems or problems into sub-phenomena, sub-systems or sub-problems with a certain intention. The greater the number of elements involved, or the less clear it is what the elements of the resulting analysis are, the more complex the analysis.

1.1 Code + documentation of a Java application

1.2 Extraction of a UML model from the code

2.1 The extracted model (given in UML diagrams)

2.2 Application of metrics onto the model, such as assessing fan-ins and fan-outs of classes in the class diagram

3.1 Characteristic dimensions of the model according to various metrics

3.2 A qualification of the stability of the application regarding extension, adaptation or usage in a different (technical) context

Dimension synthetic

Synthesizing is the combining of elements into a coherent structure which serves a certain purpose. The result can be an artifact, but also a theory, an interpretation, or a model. The greater the number of elements involved, or the more closely-knit the resulting structure, the more complex the synthesis.

1.1 A plan for a new software product (instigated by a group of prospective customers)

1.2 Making an inventory of user's requirements

2.1 User Requirements Document

2.2 Developing a functional model of the intended product

3.1 A functional model for the product (Software Requirements Document)

3.2 Developing a design for the application, taking non-functional requirements into account

4.1 The design for an application (Software Design Document)

4.2 Implementing the application

5.1 The implemented application

5.2 Deploying the application as an operational product in the environment as intended by the customers

References

Aharoni, D. (2000). Cogito, Ergo, Sum! Cognitive processes of students dealing with data structures. In Proceedings of the thirty-first SIGCSE technical symposium on Computer science education. Austin, March, pp. 26–30.

Aho, A. V., & Ullman, J. D. (1992). *Foundations of computer science*. Oxford: Computer Science.

Dale, N., & Walker, H. M. (1996). *Abstract data types—specifications, implementations, and applications*. Lexington: D.C. Heath and Company.

Haberman, B. (2004). High-school students' attitudes regarding procedural abstraction. *Education and Information Technologies, 9*(2), 131–145. doi:10.1023/B:EAIT.0000027926.99053.6f.

Haberman, B., Averbuch, H., & Ginat, D. (2005). Is it really an algorithm?—The need for explicit discourse. In Proceedings of the tenth SIGCSE Conference on Innovation and technology in Computer science education. Monte de Caparica, June, pp. 74–78.

Hammond, M., & Rogers, P. (2007). An investigation of children's conceptualisation of computers and how they work. *Education and Information Technologies, 12*, 3–15. doi:10.1007/s10639-006-9022-4.

Hazzan, O. (2002). Reducing abstraction level when learning computability concepts. Proceedings of the seventh SIGCSE Conference on Innovation and technology in Computer science education, Aarhu*s,* June, pp. 156–160.

Kaldewaij, A. (1990). *Programming: The derivation of algorithms*. London: Prentice Hall International.

Kramer, J. (2007). Is abstraction the key to computing? *Communications of the ACM, 50*(4), 36–42. doi:10.1145/1232743.1232745.

Meijers, A., van Overveld, C., & Perrenet, J. (2005). Criteria for academic bachelor's and master's curricula. Eindhoven University of Technology (revised second edition). Available online at: http://w3.tm.tue.nl/en/capaciteitsgroepen/av/platform_academic_education/

Papastergiou, M. (2005). Students' mental models of the internet and their didactical exploitation in informatics education. *Education and Information Technologies, 10*, 341–360. doi:10.1007/s10639-005-3431-7.

Perrenet, J.C., & Kaasenbrood, E. (2006). Levels of abstraction in students' understanding of the concept of algorithm: The qualitative perspective. In Proceedings of the eleventh SIGCSE Conference on Innovation and technology in Computer science education. Bologna, June, pp. 270–274.

Perrenet, J.C., Groote, J.F., & Kaasenbrood, E.J.S. (2005). Exploring students' understanding of the concept of algorithm: Levels of abstraction. In Proceedings of the tenth SIGCSE Conference on Innovation and technology in Computer science education. Monte de Caparica, June, pp. 64–68.

ACQA Project Group (2007). Onderzoek Academisch Profiel, Opleidingen BSc en MSc Informatica [In Dutch: academic profile investigation for the bachelor and master program of computer science]. Internal report, Eindhoven University of Technology.

Skemp, R. S. (1976). Instrumental understanding and relational understanding. *Mathematics Teaching, 77*, 20–26.

Tall, E., & Thomas, T. (eds). (2002). *Intelligence, learning and understanding in mathematics; a tribute to Richard Skemp*. Flaxton: Post Pressed.

van Hiele, P. M. (1986). *Structure and insight: A theory of mathematics education*. New York: Academic.

van Someren, W., Barnard, Y. F., & Sandberg, J. A. C. (1994). *The think aloud method. A practical guide to modelling cognitive processes*. London: Academic.