Levels of Abstraction in Students' Understanding of the Concept of Algorithm: the Qualitative Perspective

Jacob Perrenet
Educational Service Centre
Technische Universiteit Eindhoven P.O.Box 513,
5600MB Eindhoven The Netherlands
+31-40-2473396
j.c.perrenet@tue.nl

Eric Kaasenbrood
Department of Computer Science
Technische Universiteit Eindhoven P.O.Box 513,
5600MB Eindhoven The Netherlands
+31-628189391
e.j.s.kaasenbrood@student.tue.nl

ABSTRACT
In a former, mainly quantitative, 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. The answering level generally increased between successive year groups of Bachelor students as well as within year groups during the year, mainly from the second to the third level. The reliability of the instrument appeared to be good, but the validity remained unclear. In this current study, more qualitative methods are used to investigate the validity; the results indicate that the validity is good too. The study uses a theoretical perspective from Mathematics Education research and points at the fruitfulness of combining quantitative methods with qualitative methods.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education – *computer science education, curriculum.*

General Terms
Algorithms.

Keywords
Computer science education, research methods, abstraction.

1. INTRODUCTION
How do we know if our students are beginning to think like a computer scientist? Is it possible to distinguish abstraction levels in their thinking about core concepts? Moreover, which research methods are best suited to investigate these topics?

To answer such questions, Computer Science Education (CSE) research can build on the research experience and the theories of the closely related field of Mathematics Education [2]. In the framework of Skemp and his successors in Mathematics Education research [12] a level of abstraction has three interpretations [1], [5]:

Conference'04, Month 1–2, 2004, City, State, Country.

- Abstraction level as the quality of the relationships between the object of thought and the thinking person
- Abstraction level as a reflection of the so-called process-object duality
- Abstraction level as the degree of complexity of the concept of thought.

Just like in research described in [4] we will focus on the second interpretation: a next level is reached if a student can interpret processes and relations between objects, as a new kind of object. The process or relation becomes an object itself. If a student has reached a certain level of thinking, the lower levels still exist and are incorporated; lower levels can be evoked if necessary. As an example we will use the concept of algorithm. In [4] two levels are distinguished. "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)."

The current study is a follow up of our research reported in [9] into levels of abstraction in students' thinking about the concept of algorithm. In [9] mainly quantitative research methods were used; this paper will report about more qualitative methods. This completion is in accordance with the pledge in [3] for diversification of the research methods in CSE to both kinds of methods. Both kinds are necessary to provide a full picture of what is going on. More generally, it is in accordance with the diversification of research methods following the shift from the behaviorist to the cognitive approach in educational science (see [10] for an overview in the context of Mathematics Education).
We will start with a summary of what we accomplished in our first, mainly quantitative, study (see [9] for a detailed report) and then continue with the current, qualitative, study.

2. MEASURING LEVELS OF THINKING ABOUT THE ALGORITHM CONCEPT

2.1 Analysis of short written argumentations
We constructed a questionnaire with seven items. The starting item is worded as follows:
0. Give your definition of 'algorithm'.
The other six 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 if you cannot answer the question because of lack of knowledge, then choose 'I don't know'.* All items are followed by

the four alternatives *Agree*, *Disagree*, *Agree and disagree are possible, I don't know,* and space for supporting argumentation.

It is mainly the *supporting argumentation* that is used for further analysis, less then the choice of the answer itself. This method differs from 'normal' use of MC-questions (as e.g. in [7]). The six propositions are worded as follows:

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 generally can 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 by now.

We expected to find *four* levels of abstraction in the answers:

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.

Questionnaires were gathered several times from Bachelor students of three year groups. Table 1 gives an example of the scoring system for measuring the answering level per item.

Table 1. Detailed answer level scoring for item 1

Argumentation characteristics	Score
+ 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
Key to symbols used above: + = Agree, - = Disagree, ± = Agree and disagree, ? = I don't know. Without a clear argument no level score was given.	

2.2 Existence of levels and level growth

The main findings of our first study were as follows [9]:

- The argumentations as measured with the constructed instrument were mainly on level 2 and 3, a few on 1 or 4.
- Within a students' year group the answering level generally increased during the year.
- For successive year groups the answering level was generally higher.
- The reliability of the instrument was good (the scores given by several raters correlated well enough).

In the first study the answering levels on the various items were combined to a median student answering level. In the current study we will – for simplicity reasons - limit ourselves to answering levels per item. The results above also hold at the item level.

An important validity question remained after the first study: *did we really measure abstraction level of thinking?* The argumentations we analyzed consisted of a few lines of text. Could it not be so, that students sometimes only reproduced standard definitions, giving the false impression that they really did understand the terms used? It is this kind of question that typically asks for qualitative research, because it is the thinking *process* one is interested in.

3. STUDY METHODOLOGY

The main goal of this follow-up study was to get qualitative support for the validity of the results reported in [9]. In the first study we did an analysis on a small amount of data from many students from three Bachelor year groups; in this current study we do qualitative (deep) analysis on a large amount of data from only a few students. These students were asked to complete the questionnaire, while thinking aloud, and subsequently were interviewed about their answers. The main goal was to investigate to what extent the students really understood the computer science terms they had used in their written answers.

In Table 2 the various activities are given with an approximate time table. The mean time planned for a session was 50 minutes.

Table 2. Interview scheme with time planned

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

The reason for adding the thinking aloud task on top of the writing task, was to investigate whether this extra procedure would give

more or even different information to measure the abstraction level of the answer, or not.

The first study had given insight into what kind of answers generally could be expected, so in combination with the detailed scoring system, interview questions could be prepared (but the interviewer was free to diverge if the situation asked for it). As an example we will give some typical answers and list some questions for item 3:

The correctness of an algorithm generally can be proved by testing the implementation on cleverly selected test cases.

Answer: *Agree, exhaustive testing gives you a good idea about correctness.*
Questions:

- Exhaustive testing, how is it 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 implementation not any properties that could influence the test results were added?
- How do you know that you tested all possibilities?

Answer: *Disagree; exhaustive testing is often difficult; you should prove it.*
Or answer*: Disagree: you should prove it by using the pre/post condition technique.*
Questions:

- 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?

The interviewer should not give content information.

With some effort nine bachelor students (paid volunteers) were found willing to participate; three were in the end of their first year, two in the end of their second year and four in the end of their third year.

The sessions were taped and the protocols were typed out. The abstraction level of the answers was scored, with use of the written answers only (like in [9]) as well as with use of the extra information from the students' short thinking aloud protocols. The (longer) interview protocols 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.
- It has not become 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 counter act possible context effects (the influence of an evaluation of part of a students' work on the evaluation of another part).

4. RESULTS

Firstly we will have a short look at the scoring of the questionnaire, without and with the information of thinking aloud. Then we will come to the main part: the analysis of the students' understanding of used computer science terms.

4.1 Scoring Answering Levels

From the first part of each session we scored the answering level per student per item, one time without (the way it was done with the students in [9]) and one time *with* use of the thinking aloud protocol information. Changes caused by the extra information are only marginal. In about 85% the scores remain the same. Changes are mainly from 'score unclear' to score 2 or 3, or from score 2 to 3. In Figure 1 and 2 we give two examples of where the score changed (2 to 3). See Table 1 for the scoring system for this item. F3 means: first-year student number 3. T2 means: third-year student number 2. In these exceptional cases the extra information gave a clear indication for a higher level score.

Proposition: *An algorithm is a program, written in a programming language.*	
Written answer: *Disagree: In does not need to be in a programming language.*	level 2
Protocol: *Disagree: In does not need to be in a programming language; it can also be a specification.*	level 3

Figure 1. Written answer and thinking aloud protocol: student F3, item 1.

Proposition: *An algorithm is a program, written in a programming language.*	
Written answer: *Disagree: An algorithm can be elaborated into a program, if it lends itself well therefore, but that need not to be so.*	level 2
Protocol: *Disagree: One can implement an algorithm by means of a program, but that's only part of it, it does not hold the other way around. An algorithm is a method to solve a certain problem; an algorithm tells you – or you create it yourself – how to tackle a handle a certain problem. And you can do that by means of a certain program, but it can also by 'execute this computation on a computer' or 'throw a stone through the window'. It is all possible: one of the algorithms to open a window is 'throw a stone through the window'. If this is the right algorithm is hard to say, but it is one of them. An algorithm can be elaborated into a program, if it lends itself well therefore, but that need not to be so.*	level 3

Figure 2. Written answer and thinking aloud protocol: student T2, item 1.

4.2 Students' Understanding of Terms Used

In Table 3 the terms, as used by the students, are listed with the corresponding items and 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% it is clear that their understanding is insufficient; in about 75% of the cases their understanding of the term is evaluated as (reasonably) well. In 20% it does not get clear within the given time. Most unclearness about the students' understanding remains with the term 'complexity', like in [9], where sometimes a level score could not be given because it was not certain whether the student understood the technical meaning of the term or not.

Table 3. Terms used and extent of understanding of the terms

Terms	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[1]			?	?		?			+
pre/post condition			+					+	+
NP				+		-		?	
specification			?		+				
abstract						+			+
quantum computer				-					-
syntax							-	+	
design					+				
steps		+							
definition		+							
concept									+
formal language				+					
precise						+			
compiler				?					
natural language								+	
semantics									+
φ-scheme[2]				-					
state						+			
assembly instruct.				+					
upper bound		?							
greedy					?				
dynamic progr.ing					?				
lower bound								+	
P								+	

Key to symbols used: - The student does *not* understand the term;
? Some doubt remains 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.

While some terms are used by all the students – mainly because the terms are part of the item texts – 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. Much variation also became visible, even between students from the *same* year group, when they were interviewed about the same topic. Asked for examples of problems that cannot be solved more efficiently we observed for instance:

F1 - *The computation of the product of two numbers, when it is only allowed to use adding and subtracting, that can't 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 a NP-problem.*
T2 – *Looking up something in a telephone book. -- A computer cannot do better than binary search.*

We conclude with the two students which bring up the potential power of quantum computers, although without knowing much about it (see Table 3). While S1 more or less puts this aspect aside, T4 includes it in his 'philosophy'. See Figures 3 and 4.

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 by now.

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.* W*hen 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.*

Figure 3. Part of protocol student S1 interviewed about item 6

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 by now.

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 in stead of quantum physics, maybe there is something else; God knows how fast you are able to solve things then.*
Never say 'never'!

Figure 4. Part of protocol student T4 interviewed about item 6

[1] GCL = Guarded Command Language, a formal language for systematic construction of correct programs; see [6].

[2] φ-scheme = An algorithm design method in GCL by use of invariants.

5. CONCLUDING DISCUSSION

The first and main result of this study is, that students going through our measurement procedure for level of abstraction of thinking about the concept of algorithm generally do not give a false impression of understanding the computer science terms used by them. This strengthens the validity of the instrument. Because of the results of the first study - satisfactory reliability and the capacity to distinguish levels and to reveal level growth – both the instrument and its method of construction appear to be solid enough for further development and use. In further development a solution should be sought for the 'jargon' problem: how to measure e.g. the level of abstraction of thought about the concept 'complexity' when it is not sure whether the student knows the term in its computer science meaning. An interesting question for further research is the following one: Are levels 2 and 3 really the most common levels (which is also suggested by the results in [4]) or is level 1 relevant or even prominent for novices, e.g. pre-university students, and is level 4 prominent for experts? Concerning research into level 1, the opportunities are developing. In 2006, Master programs in Computer Science Education will start at several Dutch universities. Educational research will be integrated with these programs and concept development is one of the research topics on the agenda. In 2007, the subject of Computer Science at pre-university education, although not yet obliged, will get more volume and become more important for the students. So, the potential testees will be there too. Of course investigations should not be restricted to the core concept of algorithm alone.

The second result is that using thinking aloud protocols did not really add much for measuring the answering level with our instrument. This result says something about the *efficiency* of our instrument, more than about the *validity*. Because the thinking aloud method is very time consuming, this suggests that one should not use it without considering more efficient alternatives. Sometimes a writing task gives nearly the same information. In [8] the outline of a plan is given to let students think aloud about choosing the answers at multiple-choice questions to get more insight in the nature of their difficulties in programming. Our method of asking students to write down their argumentations for choosing one of the multiple-choice answers should be considered as a probably more efficient alternative. Besides that, the literature on the thinking aloud method – see [11] – points at the fact that the very procedure of thinking aloud can influence the thought processes that are investigated. The cause is that thinking aloud takes effort and working memory is limited. Only with relatively small tasks this influence is supposed to be negligible. But then it can be very informative.

A last point to be mentioned is the richness of data *qualitative* methods like using thinking aloud protocols and interviews can provide when combined with *quantitative* methods. Even students from the same year group showed a lot of variation when talking about the same concepts. While using closed tasks like answering multiple-choice questions, only looking at the product, could hide a lot. Dissimilar thought processes can lead to the same choice of answer and thought processes of comparable level of abstraction and understanding can lead to different answers. On the other hand, using only qualitative methods brings about a problem of generalization of the results. Only the combination of quantitative and qualitative methods can provide the whole picture.

6. ACKNOWLEDGMENTS

We would like to thank Jan-Friso Groote, the director of the Computer Science program at the Technische Universiteit Eindhoven, for making the resources to do this research available. Also we would like to thank the nine Computer Science students interviewed for their cooperation. A special thanks goes to Christina Morgan and Feline Perrenet for their helpful comments on an earlier version of this paper.

7. REFERENCES

[1] Aharoni, D. Cogito, Ergo, Sum! Cognitive Processes of Students Dealing with Data Structures. *Proceedings SIGCSE,* Austin, 2000, 26-30.

[2] Almstrum, V.L., Ginat, D., Hazzan, O. and Morley, T. Import and Export to/from Computing Science Education: The Case of Mathematics Education Research. *Proceedings ITiCSE,* Aarhuus, 2002, 193-194.

[3] Almstrum, V.L., Guzdial, M., Hazzan, O. and Peter M. Challenges to Computer Science Education Research. *Proceedings ITiCSE,* St. Louis, 2005.

[4] Haberman, B., Averbuch, H. and Ginat, D. Is it really an Algorithm – The Need for Explicit Discourse. *Proceedings ITiCSE*, Monte de Caparica, 2005, 74-78.

[5] Hazzan, O. Reducing Abstraction Level when Learning Computability Concepts. *Proceedings ITiCSE,* Aarhuus, 2002, 156-160.

[6] Kaldewaij, A. *Programming: The Derivation of Algorithms.* Prentice Hall International, UK, 1990.

[7] Lister, R. On Blooming First Year Programming, and its Blooming Assessment. *Proceedings of the Fourth Australasian Computer Education Conference,* Melbourne, 2000, 158-162.

[8] Lister, R., Fitzgerald, S., Sanders, K. and Thomas, L. *An ITiCSE 2004 Working Group: A Study of the Programming Knowledge of First-Year CS Students.* http://www-staff.it.uts.edu.au/~raymond/iticse04workinggroup/iticse04WorkingGroupV5Ray.pdf

[9] Perrenet, J.C., Groote, J.F. and Kaasenbrood, E.J.S. Exploring Students' Understanding of the Concept of Algorithm: Levels of Abstraction. *Proceedings ITiCSE*, Monte de Caparica, 2005, 64-68.

[10] Schoenfeld, A.H. (Ed.) *Cognitive Science and Mathematics Education,* Lawrence Erlbaum, London, 1987.

[11] Someren, W. van, Barnard, Y.F. and Sandberg, J.A.C.. *The Think Aloud Method. A practical guide to modeling cognitive processes.* Academic Press, London, 1994.

[12] Tall, E. & Thomas, T. (Ed.). *Intelligence, Learning and Understanding in Mathematics; a tribute to Richard Skemp*. Post Pressed, Flaxton, 2002.