Students' Conceptions of Computer Programming[†]

Michael Thuné[a] and Anna Eckerdal[b]

[a] *Email: michael.thune@it.uu.se*, [b]*Email: anna.eckerdal@it.uu.se*

[a,b]*Department of Information Technology, Uppsala University*

Abstract

The present work has its focus on university level engineering education students that do not intend to major in computer science but still have to take a mandatory programming course. Phenomenography is applied to empirical data in the form of semi-structured interviews with students who had recently taken an introductory programming course. A phenomenographic outcome space is presented, with five qualitatively different categories of description of students' ways of seeing computer programming.

Keywords: phenomenography; computer programming

[†] The phenomenographic outcome space presented in this report has previously been published as part of a journal article (Thuné and Eckerdal 2009). Due to space limitations in the journal publication, we have found it appropriate to make available a more comprehensive description of the outcome space, in the present technical report.

1. Introduction

Computer programming is a mandatory subject in many university level engineering education programs. Students experience this to be a difficult subject. It was confirmed in a big multi-national study that university students world-wide have difficulties to master computer programming (McCracken *et al.* 2001).

Against this background, there is an urgent need to find ways to improve basic education in programming, for better student learning outcome. Research in this direction has mainly focussed on students' observed abilities to understand and master various elements of computer programming. For a survey, see Robins *et al.* (2003). In those studies, the overall meaning of programming was taken for granted.

In the present paper we look at the problem from a different angle, by addressing the following question:

- *What are novice students' conceptions of computer programming?*

To address this question, we interviewed university students at the end of an introductory programming course. As a result we describe the variation in how those students express their experiences of what it means to program. Here, we will formulate this variation as a limited number of qualitatively different categories of description.

2. Variation as a key to learning and teaching

Our analysis uses *Phenomenography*, an empirically based, qualitative research approach. A phenomenographic study aims to describe the variation in the ways people experience or 'see' a phenomenon. For examples and references, see Marton and Booth (1997). The data for a phenomenographic study are typically collected via interviews with individuals. The data are analysed on a collective level, with the aim to find all the *qualitatively different* ways of seeing the phenomenon that are expressed by the interviewees. These qualitatively different ways of seeing are finally formulated by the researchers as analytical *categories of description* that form the *outcome space* of the phenomenographic analysis.

The outcome space will usually have a hierarchical structure, going from categories including few features of the phenomenon, to categories describing richer or deeper ways of seeing, encompassing several features. The crucial question for the educator is how to help students to develop rich ways of seeing the phenomenon.

In summary, via interviews we can record how students express their ways of seeing a phenomenon. As a second step, a phenomenographic analysis of the interviews will lead to

the formulation of categories of description of a limited number of qualitatively different ways in which students see the phenomenon. Below, we go through these steps for the particular phenomenon of interest in the present paper, i.e., computer programming.

3. The data collection

Our study is based on interviews with first-year students enrolled in a study programme in Aquatic and Environmental Engineering at Uppsala University, Sweden. This is a prestigious, highly theoretical four and a half year programme, leading to an M. Sc. degree. After graduation the students generally get employment as qualified specialists in their field of expertise. They are a typical example of a group of university level engineering students who are supposed to learn the basics of computer programming, but are not expected to head for future careers as programmers.

The interviews were semi-structured (Kvale 1996). This means that on one hand there was a structure in the form of 'lead questions' to open up different themes to be covered but on the other hand each lead question was open, to give the interviewee opportunities to freely express his/her understandings and experiences. The interviewer asked follow-up questions to encourage the students to talk about their experiences from different perspectives and with increasing preciseness. The interviews were conducted when the students were at the end of the introductory programming course, addressing object-oriented programming and the programming language Java. Fourteen students were interviewed. Since the objective of the interviews was to exhibit the variation in the student group, the interviewees were selected with the aim that they collectively should cover a broad range of relevant characteristics: age, gender, and previous educational, professional, and hobby experiences of computer programming. The length of an interview was typically around one hour. Each interview was recorded on tape, and then transcribed verbatim. The analysis of the data is based on the transcripts.

4. Interpretation of students' experiences

We subjected the complete set of transcribed data to a phenomenographic analysis. This is an iterative process, where the transcribed interviews are read and re-read. Statements from different interviews indicating similar ways of seeing are extracted from their contexts and grouped together, to form tentative categories of ways of seeing.

The final result of our phenomenographic analysis is a set of five categories of description of students' qualitatively different ways of seeing computer programming. These

categories of description are briefly summarised in Table 1. Note that each category of description is based on a group of excerpts extracted from different interviews. In the following sections we give an account of the categories of description and their empirical basis. The interested reader is also referred to Eckerdal (2006) and Eckerdal and Berglund (2005), where students' conceptions of *learning* to program were in focus, based on the same pool of empirical data. The categories of description reported there are very similar to the ones found in the present analysis, since students' conceptions of learning to program turned out to be strongly related to their conceptions of computer programming.

The five categories of description are organised in a hierarchy, where the categories are enumerated in increasing order of richness or depth. Each category presumes the understanding expressed in the preceding categories of description and is qualitatively different from these by including an additional feature of computer programming.

4.1 Category 1: computer programming as writing text

Category 1 describes an understanding that is directed towards the programming language, to understand its syntax. In its simplest form this means to know its key words by heart. For example, Student N[1] says:

> N: What it is all about, I think it is all about learning, partly the commands, fundamental commands I use, I have to remember them [...]

In this category, to program is understood as to write a text in a foreign language, the computer's language:

> H: […] the computer has its language, it only knows certain words. If you just write them in the correct way it will understand.

To write and read this special language is seen as non-trivial. Student E phrases this as follows:

> E: […] it is like sitting with the grammar of a language you don't know.

Student N puts it even more drastically:

> N: […] it is only nonsense, like reading old Greek.

The focus on the grammar indicates that programming is not just to write certain words, but also to use them in a syntactically correct way. Student A expresses this in saying that programming is about the following:

[1] To protect the anonymity of the informants, they are referred to by letters from A to N.

> A: […] to understand how a programming language is structured.

4.2 Category 2: computer programming as describing actions

In Category 2 there is awareness of the relation between the program text and the actions that take place when the program is executed. According to the understanding described in Category 2, the correspondence between text and action is difficult to capture and programming is regarded as an almost mystical way of thinking that can only be mastered by those who have understood this relation. In comparison to Category 1, the focus shifts from the syntax to the semantics of the program. Student D says:

> D: […] to understand the program language. That is, for example, to be able to look at a program and see that, okay, this will happen and this is what the program will do […]

To program is now experienced as to think in a particular way required in order to write a program that performs the intended actions when executed:

> E: Well, it is precisely to "think programming", I believe. To understand things, put them together, and how you get things to work, like, and do what you want.

This particular way of thinking is seen to differ from "ordinary" thinking in that it must be very precise. This is because the computer does not have the human ability to fill in missing, "evident" details. So, according to this understanding, programming is to think through all the steps that the computer must carry out in order to achieve the desired actions, and then to write down a syntactically correct description of those steps in the programming language:

> C: […] you have to be very exact in everything you describe to the computer that you want it to do. It is not so open to interpretation. So that, well, then you have to think it through, yourself, what you want it to do.

4.3 Category 3: computer programming as producing tools for use in everyday life

In Category 3, the perspective is widened to include not only that the program is intended to do something, but that the things to be done are related to real applications that you encounter in everyday life. The following excerpts from interviews illustrate this:

> B: […] for me it has been about understanding how things are constructed. That is to say that I've understood things, how bank programs work and such […]

> C: Well, it is in cars, computers, elevators, and everything.

> D: You only think of things like when you are to extract money from an ATM […]

4.4 Category 4: computer programming as a problem solving

The understanding described in Category 4 includes the same features of programming as the previous three categories, but now the understanding of the relation between text and action is clearer and the thinking required to program is seen as a kind of systematic problem solving:

> F: […] to think in a certain way […] problem solving
>
> C: […] I guess it's actually to solve a certain type of problem […] different methods to solve them in different ways.

Student K expresses the view that the program is a result of a systematic problem solving process:

> K: You have a problem that you solve in different ways and then you maybe find the best way. That is one of the central parts I think. Then, that you should write in some kind of programming language, you could maybe do that in any programming language. But precisely to handle problems, to handle problem solving, that is what I consider as important.

Here, the programming language is in the background and focus is on a higher level of abstraction where a program is not primarily considered as a text in a certain language but as an expression of a set of algorithms:

> D: […] how you, like, pose these problems and how you solve certain problems and what kinds of algorithms, sort of, that you should formulate […]

4.5 Category 5: computer programming as an empowering skill in various contexts

Finally, Category 5 includes all the previous categories and adds the insight that knowledge in computer programming can be useful outside the programming course and for other purposes than programming. Student C says that knowing about programming can be useful because it helps you to understand how many devices work:

> C: It will maybe be useful now and then […] understanding how appliances work in general, and machines.

In particular, to understand how computers work will be useful in the work place, regardless of what profession you end up in, says Student A:

> A: I find it difficult to think that there is any job where, that is the impression I get, where computers are not involved in some way. So it is really good [to have an] introduction and understanding of that.

To have this broad understanding will be empowering in the sense that it makes you less dependent on others. Student E expresses this as follows:

> E: Yes but it's that the more you know about computers the less dependent on others you'll be, sort of.

> E: […] if you work somewhere later on and have a bit of insight into things, then I think that will open a small window […]

5. Related work

Other researchers in the phenomenographic tradition have addressed similar questions as the ones posed in our study. In particular, Table 1 points in the same direction as the outcome spaces reported by Booth (1992), and Bruce *et al.* (2004), respectively. As part of a larger study, Booth investigated students' conceptions of learning to program. Her outcome space has four categories, three of which are similar to our Categories 1, 2 and 4. Her fourth category, 'learning to program as becoming part of the programming community' does not appear in our data. Most likely, this is due to the fact that Booth interviewed computer science majors, while the students in our study were not heading for careers in the software industry. This difference is probably also the reason why our Category 3 did not appear in Booth's outcome space. Similar comments can be made with regard to the study by Bruce *et al.* (2004). We do not go into a detailed comparison here. The important message is that Table 1 points in the same direction as the outcome spaces by Booth, and Bruce et al., respectively. These three outcome spaces paint the same basic picture, while each outcome space adds partly different nuances to it.

Table 1 represents the lived object of learning (Marton *et al.* 2004), that is, how the students actually were able to see computer programming after having taken an introductory university course on that subject. From an educational point of view, it is of interest how those categories of understanding relate to the intended object of learning (Marton *et al.* 2004), as conceived by educators. An authoritative source of information concerning the intended object of learning in introductory computer programming courses is provided by *Computing Curricula 2001—Computer Science* (CC 2001, see Engel and Roberts 2001). It is implied in CC 2001, Chapter 7.2, that a desirable goal for an introductory programming

course is that students attain the ability to "adapt to different kinds of problems and problem-solving contexts" in the future. This corresponds fairly well to the understanding of computer programming described in Category 5, keeping in mind the inclusive character of the five categories of description.

6. Conclusions

To get a nuanced understanding of novice students' conceptions of computer programming, we collected data via interviews at the end of an introductory programming course. A phenomenographic analysis of these data resulted in the categories of description summarised in Table 1. The outcome space in Table 1 demonstrates a wide span between different ways of seeing computer programming in a group of novice programmers who had taken the same introductory programming course. In the least advanced categories of description very few features of computer programming are observed. The more advanced categories relate to several features of this phenomenon.

The educational challenge is to create learning conditions that enable students to see additional features of computer programming. According to *variation theory* (Marton and Tsui, eds, 2004), creating such learning conditions means to create possibilities for students to experience variation in dimensions related to the features. Experiencing such variation is a necessary, though not sufficient condition for discerning the features.

In a separate publication (Thuné and Eckerdal 2009), we have applied variation theory to the phenomenographic outcome space presented above. There, we also discuss implications for teaching based on the results of the combined phenomenographic and subsequent variation theoretical analysis.

References

Booth, S. A., 1992. *Learning to program*. Thesis (PhD). Gothenburg University, Sweden.

Bruce, C., Buckingham, L. Hynd, J., McMahon, C., Roggenkamp, M., and Stoodley, I. 2004. Ways of experiencing the act of learning to program. *Journal of Information Technology Education*, 3, 143–160.

Eckerdal, A. and Berglund, A., 2005. What Does It Take to Learn 'Programming Thinking'? *In*: *Proceedings of the 1st International Computing Education Research Workshop*. New York: ACM Press,135–143.

Engel, G. and Roberts, E., eds., 2001. Final Report on Computing Curricula 2001—Computer Science. *ACM Journal of Educational Resources in Computing*, 1(3), 1–240

Lo, M. L., Marton, F., Pang, M. F., and Pong, W. Y., 2004. Toward a Pedagogy of Learning. *In*: F. Marton and A. B. M. Tsui eds. *Classroom Discourse and the Space of learning*. Mahwah: Lawrence Erlbaum, 189-225.Marton, F. and Booth, S. A., 1997. *Learning and awareness*. Mahwah, NJ: Lawrence Erlbaum Associates.

Marton, F., Runesson, U., and Tsui, A. B. M., 2004. The space of learning. *In:* Marton and Tsui (2004), 3–40.

Marton, F. and Tsui, A. B. M., eds., 2004. *Classroom discourse and the space of learning.* Mahwah, NJ: Lawrence Erlbaum Associates.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., and Wilusz, T., 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *SIGCSE Bulletin* 33(4), 125–180.

Robins, A., Rountree, J., and Rountree, N., 2003. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2), 137–172.

Thuné, M., Eckerdal, A., 2009. Variation theory applied to students' conceptions of computer programming. *European Journal of Engineering Education*, 34(4), 339–347

1. Computer programming is experienced as to use some programming language for writing program texts.
2. Computer programming is seen a way of thinking that relates instructions in the programming language to what will happen when the program is executed.
3. Computer programming is seen as a way of thinking, as above, and in addition computer programming is experienced as producing computer programs such as those that appear in everyday life.
4. Computer programming is seen as described above with the addition that computer programming is experienced as a "method" of reasoning that enables problem solving.
5. Computer programming is seen as a way of thinking, to solve problems, leading to the production of computer programs such as those that appear in everyday life. In addition computer programming is experienced as a skill that can be used outside the programming course, and for other purposes than computer programming.

Table 1: Summary of categories of description of students' qualitatively different ways of experiencing computer programming