Towards an experiential description of programming and learning to program.

Shirley Booth
Department of Education and Educational Research
University of Gothenburg
Sweden

shirley@ped.gu.se

Abstract

The work I wish to present is a move towards describing learning to program (as a limited aspect of programming in a more general sense) by taking an experiential perspective on programming. By "an experiential perspective on programming" I mean focusing on what occurs when a person (a student or an experienced programmer) meets a problem for which a program has to be devised, and what shapes the program that emerges. That is to say, it is the programmer's experience of writing the program that is to be described, in terms of his or her awareness of the problem and the resulting developing field of awareness of the required program. In contrast to a purely psychological perspective, this experiential perspective aims neither to determine possible rules of programming behaviour, nor to posit structures and forms of programming knowledge, nor to explain the programming process in a normative sense. The experiential approach to describing programming seeks rather to investigate the relationship between the programmer, the problem and the evolving program, and tries to answer the question, "What does it mean, to learn to program?"

Such an experiential perspective has been discussed by Peter Naur, who took a keen interest in programming education and programming behaviour; in 1965 he wrote as follows about programming:

> ... we are dealing with three elements: problems, tools and people, and [...] the essence of the situation is the interplay between them. More specifically, we cannot, as people, think of a problem without at the same time implying some kind of tool. Stronger yet, when the tool changes, the problem is not the same anymore. On the other hand, our opinion about what is a proper, or desirable, tool surely depends on our understanding of the problem. In any case the tool and the problem are nothing if they are not recognized as such by a person– that is where the people come in. (Naur, 1965, p 3)

In 1985, stating at the same time that he would "use the word programming to denote the sole activity of design and implementation of programmed solutions", he wrote:

> It is suggested that programming properly should be regarded as an activity by which the programmers form or achieve a certain kind of insight, a theory, of the matters at hand. This suggestion is in contrast with what appears to be a more common notion, that programming should be regarded as a production of a program and certain other texts. (Naur, 1985, p 37)

Instead of taking a psychological perspective, looking for generalities, he was taking more of an experiential stance in talking of the meeting between the programmer and the problem for which a program has to be written and the development that ensues.

The paper is based on an empirical study of university students learning to program in a functional programming language, SML (Wikström, 1987), carried out with a phenomenographic approach. A phenomenographic study aims to describe qualitatively

distinct ways in which people experience, or understand, or conceptualise particular phenomena of interest (Booth, 1992a). In particular, such studies are carried out in educational settings, one goal being to inform teachers of ways in which their pupils or students are probably understanding the ideas (or phenomena) they are meeting. The phenomenon in question may be either central and thematised (e.g. recursion in a study of student programmers (Booth, 1992b)) or peripheral to instruction, unthematized, yet still of importance (e.g. the nature of programming and programming languages (Booth, 1990)). An extension of phenomenographic studies is sometimes to delve into the structure of topics of study, seen through the eyes or awareness of the learners, with the goal of making didactic changes which can lead to better understanding. Apart from phenomena characterised as the central themes of instruction (e.g. recursion, program correctness, functions) and phenomena more peripheral to learning to program, this study also focused on students tackling specific text-book problems, and it is this part of the study which is central to the paper.

In the study, 14 undergraduates studying Computer Science and Computer Engineering, chosen to represent the class variation of factors such as age, gender and educational background, were interviewed six times during their initial course in programming. The interviews were – following phenomenographic principles – semi-structured, open and deep. This means that while certain themes were planned in advance to form the structure of each interview in a set (semi-structured), after the planned opening questions the discussion was allowed to take a natural course (open) in an effort to understand the student's thinking on the theme (deep). Some themes were taken up in more than one interview and other themes were taken up when topical. Soon after the start of their studies, students were asked, in an interview situation, to try a very small and apparently simple text-book problem – the *take* problem (Fig. 1) – which was taken from their current problem sheet. I am not suggesting that this was an interesting problem in itself, nor is it like the problems the students have to face in laboratory projects, let alone in the future work-place. It was, however, a challenge for these students who were just starting to develop algorithms and express them as programs in the unfamiliar SML functional form.

Write a function *take* which has two arguments: an integer *i* between 1 and 8 and another integer *n* between 1 and 99999999. The function is to return the *i*th digit of *n*. For example:

```
- take (1,1234);
> 4:int
- take (2,1234);
> 3:int
- take (3,1234);
> 2:int
- take (4,1234);
> 1:int
```

If there is no *i*th digit in *n* then a zero should be returned:

```
- take (5,1234);
> 0:int
```

Figure 1. The *take* problem

They were asked to read the question and then tell me what it meant. Then they were asked to try to write a suitable program; I let them work as they wished but requested that they should describe what they were doing and broke in on occasion for clarification. When a solution was obtained or impasse reached, the work undertaken was discussed with the aim of reaching an understanding of what had been done and what the result of the algorithm might be.

The data thus obtained was analysed together with data from a later and similarly conducted program writing session. The main result consists of a set of four qualitatively distinct approaches to writing programs. I use the word "*approach*" to refer to the way in which the student immediately identifies what is central to the problem in hand, which then constitutes the starting point for the development of a program. These have been named

- an *expedient approach*, where a program or function from SML's standard repertoire is the starting point;
- a *constructual approach*, where possible constructs needed for the program are the immediate focus;
- an *operational approach*, where the operations needed within the program are taken up as a start; and
- a *structural approach*, where it is the structure of the problem in its own domain that is first considered.

The four identified approaches will be described more thoroughly and discussed in the paper, but it can be pointed out that, in the order they are stated above, they form a hierarchy of greater meaning for the inexperienced student. The first pair tend to opportunism while the second pair are interpretative, and while either of the first two are common and to some extent desirable in professional programming, students generally lack the experience which leads to their being successful.

Taking these empirically obtained and logically distinct *approaches* as a point of departure, I am now looking at their significance for learning to program in terms of the ways in which students were found to understand other phenomena connected with programming, both technical and more peripheral, much in the ways suggested by Naur. The ideas of the field of awareness and the stream of awareness which were developed by the phenomenologist Aron Gurwitsch (Gurwitsch, 1964) have laid the ground for an experiential description of programming and learning to program, which comes some way toward integrating the results of the study in that it enables the interplay between different ways of understanding different aspects and tools of programming to be discussed. My immediate aim is, on the one hand, to elaborate and refine this description – by drawing on the results of the existing study, by recourse to other studies, and by planning further studies – and, on the other hand, to consider mutual implications for psychological research into programming and learning to program. In the longer term, one aim is to develop the phenomenographic, or phenomenological, view of learning complex subjects such as programming, where logical or rational thought and problem solving play an important role, and a second – but equally important – aim is to develop the didactic implications of such a view of learning.

References

Booth, S. A., (1990). *Conceptions of programming: A study into learning to program.* Publications of the Department of Education and Educational Research of the University of Gothenburg. 1990:01.

Booth, S. A. (1992a). *Learning to program: A phenomenographic perspective.* (Göteborg studies in educational sciences 89). Göteborg: Acta Universitatis Gothoburgensis

Booth, S. A. (1992b) The experience of learning to program. Example: Recursion. In F. Détienne (Ed.), *5 ème workshop sur la psychologie de la programmation (The fifth workshop of the "Psychology of Programming Interest Group")*, (pp 122-145). Paris: INRIA.

Gurwitsch, A. (1964). *The field of consciousness.* Pittsburgh: Duquesne University Press.

Naur, P. (1965). *The place of programming in a world of problems, tools and people.* Proc. IFIP Congress '65, pp 195-199. Reprinted in P. Naur, (1992). *Computing: A human activity.* New York: ACM Press.

Naur, P. (1985). *Programming as theory building.* Microprocessing and microprogramming, 15, pp 253-261. Reprinted in P. Naur, (1992). *Computing: A human activity.* New York: ACM Press.

Wikström, Å. (1987). Functional programming using Standard ML. Hemel Hempstead: Prentice Hall.