THE ROLE OF METAPHORS IN NOVICES LEARNING PROGRAMMING

Ann Jones

The Open University
Milton Keynes, England

Abstract

Learning a complex skill such as programming requires the developments and use of conceptual models, both of the concepts in the programming language, and the 'behaviour' of the machine. The latter has been referred to as the 'notional machine' (du Boulay, B., O'Shea, T. and Monk, 1981). Such a conceptual model, however, must interact and build upon models and metaphors which students already have. It is these metaphors and some techniques for studying them which are discussed in this paper.

Introduction and Background

Behavioural studies of programming are motivated by a diversity of goals, for example a desire to understand the task better and thus how it can be performed more efficiently: or a concern about the importance of developing procedural literacy, (for example, Sheil, 1980, (b)) or an interest in programming as an applied example of a high level skill. It is the latter, mainly, which motivates the present study: the overall question, although it is far from simple, can be simply phrased 'What goes on in the mind of the learner programmer?'

There is now a substantial body of research on programming, although Sheil, (1980) argues that many empirical studies of programming have added very little to our knowledge of what it means to learn programming, partly because the methodology is fraught with difficulties but mainly because we still know so little about what the task entails: how programming knowledge is organized and how it can be represented. There is some agreement that it can be thought of as a collection of units, (or 'frames', 'paradigms' or 'schemas') organised as program fragments with a set of propositions about its behaviour and rules for combining it with others. (Rich 1978, Floyd, 1979.) There is no evidence, however, that novices have access to such structures; on the contrary, studies of the knowledge organization of experts and novices, in the programming domain, indicate not unsurprisingly, that novices lack such organizing schemas. (McKeithen and Reitman, 1981; Adelson 1981). It has been argued (du Boulay, O'Shea and Monk, 1981.) that one of the difficulties of teaching a novice programming is how to describe at the right level of detail the machine she is learning to control; and as a way of doing this they suggest teaching using the idea of a notional machine - an idealised conceptual computer whose properties are implied by the constructs in the programming language employed. The notional machine is similar to the 'transactions' which Mayer uses to describe the workings of a BASIC machine (Mayer, 1979), and also suggests as a basis for teaching BASIC. Other studies, (e.g. Miller, 1974) have also emphasized providing novices with carefully thought out metaphors and models to help them learn. The gap, however, in all this is what the learner herself brings to the situation, and that is, all her past experience and knowledge which will be used to interpret and organize the new material that is to be learnt. Although programming has many specialised terms, many words do have everyday meanings and associations which are different from their programming use and may not facilitate the learning process. Such words will not necessarily have a shared meaning among novices. McKeithen and Reitman (1981), in studying the organization of programming knowledge found that beginners' organizations show a rich variety of common language associations to these programming concepts. Botts' study of learning how to use a text editor, (Bott, 1979) suggests that such pre-associations are powerful and pervasive, and may not be easily replaced by new metaphors and models.

The Study

The study of novice programmers' initial metaphors and models is being investigated as part of a larger study of how students learn two very different programming languages: only one of which will be discussed here, a language called SOLO. The students using SOLO are taking an Open University Cognitive Psychology course; and SOLO provides an environment for them to manipulate an assertional data base, as a tool for learning and thinking about knowledge representation. It was designed to make life easy as possible for total novices by being restricted to a small number of primitives, and incorporating many user aids, such as a spelling corrector. Nevertheless, learning to program in SOLO is by no means trivial, and so it has been necessary to find out what problems the students have so that the programming environment can be tailored to suit their needs. (Eisenstadt, Laubsch and Kahrey, 1981). In order to build a really fool-proof environment it is necessary to understand precisely what the novice really thinks is going on inside the SOLO machine. Of course, trying to find out what someone 'really thinks' is going on is a tall order, and to a large extent this study has been concerned with exploring methodologies which will provide 'windows' into novices' conceptual models of the SOLO machine. Several such windows have been explored: students were asked to actually start learning SOLO in the laboratory and to talk aloud while doing the exercises in the primer, (for this study, the fact that SOLO is learnt at a distance, from a correspondence text, is a great advantage, as the teaching method and techniques used are made explicit and are consistent across students); they are interviewed and asked to talk about some of the concepts before they started; they completed Repertory grids, (Kelly, 1970) and the worked through some very simple exercises. The next section will briefly discuss the virtues and problems of some methodologies for specifically exploring the metaphors and models which students bring with them, and some examples of these.

The Repertory grid

Repertory grids are usually used to elicit constructs rather more general than those concepts used in programming. They are a way of finding out how a person categorizes the world or some part of the world. Other studies of knowledge structures, e.g. Adelson (1981) have investigated how programs

are organised. The approach used here however was to ask students to categorize the actual promitives of the SOLO language, and subjects talked aloud while they did the exercise. In this exercise they were shown three of the SOLO 'words' on cards, and asked if there was a way in which two were alike and one was dissimilar; and told that they would then categorize the rest of the cards according to this construct. This is repeated until no more constructs can be elicited. In doing this I was interested in the constructs a subject would choose, given the freedom to carve up the 'SOLO world' however she wanted, and also whether, given a construct elicited from the first triple it would be possible to categorize the remaining SOLO terms in accordance with it. For example, one subject started with NOTE (an 'Assert' function), CHECK (a 'retrieve' function) and LIST - which lists procedures. She said that NOTE was to do with 'Giving (the database) new things' whereas CHECK and LIST have a retrieval function - 'they show you what's there.' It's not clear how much sense such constructs have for the rest of the primitives. This subject's categorization for this construct was:

"Giving new things"	"Retrieval"
DESCRIBE	IF PRESENT
NOTE	IF ABSENT
To (Defining a procedure)	CHECK
CONTINUE	EDIT
EXIT	
PRINT	
PARAMETER	
VARIABLE	
FORGET (delete)	

In this case, the construct does make some sense as a way of categorizing the SOLO terms. As might be expected from other related studies (eg Adelson, 1981) many of these beginner programmers used constructs which were idiosyncratic and related to everyday knowledge as much as to programming knowledge.

Eliciting such constructs is clearly a hard activity for beginners (as indeed it is for experts), but although subjects found it demanding they also found it rewarding as it forced them to think about how they organized these concepts which they were not aware of having categorised. This thinking it through becomes explicit in their verbalization. What seems to be happening to this is that the students are learning while doing the task. Whilst this is exciting and can provide rich data it is also clearly problematic for analysis as the learner's state is changing during the exercise.

Secondly, some students seem to "chain" concepts; to lose track of the construct or category and to categorize each element or concepts (the primitives) according to its similarity or otherwise, to the previous one.

The grouping of the constructs for each subject and differences between subjects has not yet been analysed, and so, overall, it is difficult at the moment to assess how useful this technique will be.

Concept Interviews and Talking through Exercises

The most fruitful technique so far for investigating metaphors has been a combination of concept interviews and students talking aloud while working through the exercises in the promer. In the 'concept' interviews, subjects were asked to talk about some of the words used in the SOLO language before starting to learn about it. Information about how a student interprets a certain word before even starting provides a baseline for interpreting their behaviour. Some interpretations are remarkably common, yet unsurprising, (in retrospect!) "Parameter", if it elicited any reply at all, was "limits", "boundaries" "constraints", which is not particularly close to its programming use. 'Node', on the other hand, which is a word used in SOLO's semantic network, was for many the biological 'node' of a nerve network; - a not inappropriate metaphor.

Protocols of students working through exercises in the teaching test were also taken. Consider the following extract from a transcript, which conerns how procedures take paramenters:

"I think of it in relation to a sort of work processor, that if I was doing a lot of letters I would do a letter and put an X in 'Dear X' and then each one I'd just print in Fred, Mary.. so that each letter...."

This metaphor is very useful for a novice thinking about the idea of a procedure taking a parameter. It should be stressed however that this is something the learner brings with her; no matter what metaphors are offered in the teaching, the student must map what she's learning on to her own experience. Normally this is a 'hidden' activity; but part of the aim of this research is to make it 'explicit'. Such information can pay dividends later, ss such metaphors may only be useful for a fragment of time, or for learning on particular thing. In the above example, the metaphor led to expectations about the way the editor would behave, - that it would be able to delete single words within a line of the procedure, which did not match its actual behaviour, as in fact, whole lines must be deleted, and yet, such beliefs were persistent. Both (1978) cites the way in which such expectations can lead to interpretations which are false, yet very pervasive, and how complicated stories are constructed to account for the mismatch between expectation and reality, before the learner finally discard an inappropriate schema.

Conclusions

In helping novices to learn programming, it is not enough to provide metaphors and models. In addition we must study the models and metaphors students already have, and which they bring with them to the learning situation. This paper has discussed such metaphors and possible methodologies for investigating them. The next step is to examine how existing metaphors interact with experience in the development of conceptual modes of a programming language. This is a vital issue in understanding how novices learn programming.

Acknowledgements

I would like to thank Tim O'Shea and Richard Young for help and advice on this paper, and for helpful discussions.

References

Adelson, B. Knowledge Structures of Computer Programmers Proceedings of 3rd Annual Conference of Cognitive Science Society, 1981, pp 243-248.

Bott, R.A. A Study of Complex Learning. Report No. 82, University of California, Centre for

Human Information Processing, 1979.

du Boulay, B., O'Shea, T., and Monk, J. The black box inside the glass box: presenting computing concepts to novices. Int J Man Mach Studies, 1981, 14, 237-249.

Eisenstadt, M. Artificial Intelligence Project, Cognitive Psychology Course, Open University 1978.

Eisenstadt, M., Laubsch, J., and Kahney, H. Creating Pleasant Programming Environments for Cognitive Science Students. Proceedings of 3rd Annual Conference of Cognitive Science Society 1981.

Floyd, R,W, "The Paradigms of Programming" Commus. ACM 228 (August 1979) 455-460.

Kelly, G.A. A brief introduction to personal construct theory. In Perspectives in Personal Construct Theory (BANNISTER D. Ed), London Academic Press, 1970.

Mayer, R. A Psychology of Learning BASIC Communication of the ACM, Vol. 22, No. 11, Nov. 1979.

McKeithen, D.B. and Reitman, J.S. et al. Knowledge Organization and Skill Differences in Computer Programmers. Cognitive Psychology, 13, 307-325, 1981.

Miller, L.A. Programming by non programmers. Int J Man-Mach Studies, 1974, Vol. 6, 273-260.

Rich, C., and Shrobe, H. "Initial report on a Lisp programmer's apprentice", IEEE trnas. Softw. Eng. SE-4 (1978) 456-467.

Sheil, B. The Psychological Study of Programming, Computing Surveys, Vol. 13, Nol. 1, March 1980.

Sheil, B(b). "Teaching procedural literacy" in Proc. ACM Annual Conf. 1980, pp 125-126.