Structural knowledge and language notational properties in program comprehension

Pablo Romero and Benedict du Boulay
Human Centred Technology Group,
University of Sussex, Falmer, BN1 9QH, UK.
pablor@sussex.ac.uk

Abstract

Several accounts of program comprehension have taken the theory of text comprehension by Kinstch as a starting point to model the mental representations built when programmers understand a computer program. A crucial point that these accounts try to explain is how these mental representations are organised. According to Kintsch's theory, the mental representations built as a product of the text comprehension process are interrelated propositional networks whose organisation is determined by the main idea of the text. In program comprehension, this main idea has been understood in terms of functionality. This paper contends this notion, proposing that in program understanding programmer's mental representations are multifaceted and organised through several criteria. Which of these is the most important one depends on the programming language employed among other factors. The fact that functional information appeared as crucial might have been because most of the empirical research that has been undertaken has employed procedural languages. This claim is tested empirically by analysing the mental representations of programmers in Prolog, a declarative programming language. The results support our claim by showing that in this case data structure information is more important than function.

Keywords: knowledge representation, program debugging, program comprehension.

1 The organisation of programmers' mental representations

There is a tradition in program comprehension studies to regard this cognitive activity as similar to text comprehension. A number of studies have taken methods and theories of text comprehension as a starting point to build models of program comprehension [1, 14, 6, 5]. In particular, the text comprehension theory developed by Kinstch [12, 11] has inspired most of these models. One of the aspects of program comprehension that has been frequently compared to text comprehension is the mental representation that programmers are said to build of the programs they study. This mental representation is said to be organised into a hierarchical structure which is dictated by the situation being described by the text as well as by the reader's general knowledge about the state of affairs in the world [10, 12, 11]. This hierarchical structure makes topic sentences of a paragraph more salient or relevant for readers [11]. In program comprehension, this main idea of a paragraph has been equated to the place in the code where the main function of the program is made explicit [4, 17, 15, 5]. It is considered that through experience, programmers develop the necessary skills to detect these important places in the program. However this research has considered procedural languages mainly; it is not clear that this is true for languages belonging to other paradigms.

Research that has considered procedural languages has identified the most important places of the code as the focal elements of Programming Plans [15, 5]. Programming Plans can be described as units of schematic stereotypical programming knowledge related to the program's goals. Studies that have taken into account languages belonging to other programming paradigms sometimes have found Programming Plans unable to explain some aspects of programmers' knowledge [9, 2, 13]. It has been argued in [16] that although Programming Plans might be important for non-procedural languages, there might be other valid models for them. For a declarative language like Prolog, for example, Prolog Schemas [7], a model related to data structure information, has been proposed as a plausible alternative [16]. These studies therefore suggest that although programmers' mental representations are hierarchically organised and some elements of this organisation are focal, the criteria behind this hierarchical organisation might not always be based on functional information. Programming languages, as any other information structures, highlight certain types of information but obscure some others [8].

The information type(s) highlighted seem to be the ones that dictate the criteria for the hierarchical organisation of the programmers' mental representations. This suggests that although program comprehension is similar to text comprehension in that mental representations are hierarchically organised and some elements of this organisation are focal, in program understanding programmers' mental representations are multifaceted and organised through several criteria. Which of these is the most important one depends on the programming language employed among other factors. The following section describes an empirical study designed to test this hypothesis.

2 Comparing two structural models through a debugging task in Prolog

In the empirical study reported in this section two groups of programmers, novice and experienced Prolog users, performed a debugging task on several Prolog programs each. These programs were seeded with errors located in places of the code i) considered as focal according to a Programming Plans analysis (*plan key errors*), ii) considered as focal according to a Prolog Schemas analysis (*Schema key errors*) and iii) somewhere else (*non-key errors*). The detection time and accuracy of the two experimental groups were compared to find out which kind of error was more easily spotted by which group. The hypothesis proposed above implies that, as Prolog highlights data structure information [16], the mental representations of experienced Prolog programmers will be organised hierarchically according to a data structure criterion. Therefore it is expected that places of the code considered as focal according to a data structure analysis will be more salient for experienced programmers, and consequently deviations (errors) in these segments would stand out more than deviations elsewhere.

2.1 Design

This experiment was a 2 $\times$ 3 factor design with two independent variables (skill level and error category) and one dependent variable (error detection accuracy). Skill level was a between subjects variable and error category a within subjects variable. The skill levels were novice and experienced Prolog user and the error categories were Schema key errors, Plan key errors and non-key errors.

2.2 Participants and Procedure

The participants of the experiment were 14 novice and 12 experienced Prolog programmers. The novice population comprised undergraduate and postgraduate (MSc) students who had taken a three month introductory course in Prolog and were either Computer Science and Artificial Intelligence undergraduates or Artificial Intelligence master students, all of them in the School of Cognitive and Computing Sciences in Sussex University. The Computer Science and Artificial Intelligence curriculum normally includes the teaching of one or two programming languages before Prolog, therefore, this novice population was inexperienced in Prolog, but not in programming in general. They knew at least another programming language already.

The experienced programmer population had on average 9.6 years of Prolog programming experience and had written programs longer than one thousand lines (on average). They were either lecturers or researchers at academic institutions and five of them had taught a Prolog course.

The experiment was a pen and paper exercise and the subjects performed eight debugging sessions. In each one of them, they debugged a different Prolog program. First they had a period of time to read the program specification. After this, they were presented with the program code. Their task at this point was to verify that this code complied with its specification. If this was not so, they had to mark any possible problems, but not to correct them. They had up to 5 minutes to perform this task. Each program was seeded with three errors, one Plan key, one Schema key and one non-key. The order of presentation of these programs was randomised.

2.3 Results

The statistical analysis compared debugging accuracy (detection rates) through a repeated measures ANOVA for skill level as between subjects condition and error category as within subjects variable. The results of this analysis are summarised in Figure 1. It can be seen that although experienced subjects had a higher detection accuracy than novices for every error category, Schema key errors stand out as those for which this difference is the largest. There were main effects for skill level, F(1,24) = 27.78, $p < .01$; and for error category, F(2,48) = 7.86, $p < .01$. There was also an interaction effect for these two factors, F(2,48) = 4.54, $p < .05$. Tests of interaction effects contrasts were run comparing a) Schema vs. both Plan and non-key errors and b) Plan vs. non-key errors. The only comparison with a significant difference was the first one, F(1,24) = 8.46, $p < .05$. This result confirms the tendency shown in Figure 1 and commented on above about the large difference between the accuracy rates of experienced and novice subjects only for the case of Prolog Schemas.

This results suggest that, at least for these experimental conditions, the mental representations of experienced Prolog programmers, unlike those of novice users, display a hierarchical organisation based on data structure. This is different from research that has considered procedural languages [4, 17, 15, 5] (which suggests an organisation

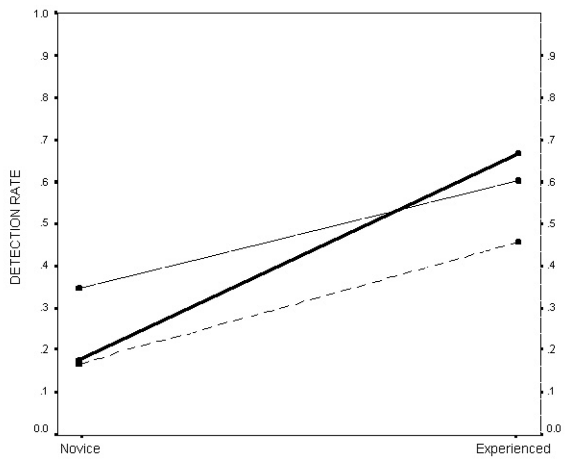

Figure 1. Error detection accuracy rate by novices and experienced programmers.

(——) Plan key errors; (**——**) Schema key errors; (– – –) non-key errors.

of mental representations in terms of functional aspects) and similar to research that has focused on Prolog [16]. This result supports the experimental hypothesis which suggested that although program comprehension is similar to text comprehension in that mental representations are hierarchically organised and some elements of this organisation are focal, this hierarchical organisation is not always in terms of function; the target programming language plays a crucial role in determining the criteria for this organisation. Taking into account this result together with other findings in the area [3, 16], instead of a mental representation dictated by only one structural model, the picture that emerges here is of a multifaceted mental representation perhaps organised through several criteria. This finding is especially important for programming languages other than procedural ones. It cannot be assumed, for example, that the mental representations of programmers of Object Oriented programming languages will be organised according to a functional criterion. Research taking into account this sort of languages has to be undertaken to determine which is the case here.

This paper has described an empirical study designed to test the hypothesis that programmers' mental representations are multifaceted and can be organised through several criteria and not just in terms of functional aspects. As the target programming language is an important factor to establish this criteria more research which takes into account non-procedural programming languages is needed in this area.

References

[1] M. E. Atwood and H. R. Ramsey. Cognitive structures in the comprehension and memory of computer programs: an investigation of computer program debugging. Technical Report TR-78-A21, US Army Research Institute for the Behavioral and Social Sciences, Va: Alexandra, 1978.

[2] R. K. E. Bellamy and D. J. Gilmore. Programming plans: Internal and external structures. In K. Gilhooly, M. T. G. Keane, R. H. Logie, and G. Erdos, editors, *Lines of thinking: Reflections on the psychology of thought, Vol 1*, pages 59–71. Wiley, London, U.K., 1990.

[3] D. Bergantz and J. Hassell. Information relationships in PROLOG programs: how do programmers comprehend functionality? *International Journal of Man-Machine Studies*, 35:313–328, 1991.

[4] R. Brooks. Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies*, 18:543–554, 1983.

[5] S. P. Davies. Models and theories of programming strategy. *International Journal of Man-Machine Studies*, 39:237–267, 1993.

[6] F. Détienne. Expert programming knowledge: a schema-based approach. In J. Hoc, T. R. G. Green, R. Samurçay, and D. J. Gilmore, editors, *Psychology of Programming*, pages 205–222. Academic Press, London, U.K., 1990.

[7] T. S. Gegg-Harrison. Learning Prolog in a schema-based environment. *Instructional Science*, 20:173–192, 1991.

[8] D. J. Gilmore and T. R. G. Green. Comprehension and recall of miniature programs. *International Journal of Man-Machine Studies*, 21(1):31–48, 1984.

[9] D. J. Gilmore and T. R. G. Green. Programming plans and programming expertise. *Quarterly Journal of Experimental Psychology*, 40A:423–442, 1988.

[10] W. Kintsch. On comprehending stories. In M. A. Just and P. Carpenter, editors, *Cognitive Processes in Comprehension*, pages 33–61. Erlbaum, Hillsdale, NJ, 1977.

[11] W. Kintsch. *Comprehension: a Paradigm for Cognition*. Cambridge University Press, Cambridge, U.K., 1998.

[12] W. Kintsch and T. A. van Dijk. Toward a model of text comprehension and production. *Psychological Review*, 85(5):363–394, 1978.

[13] T. C. Ormerod and L. J. Ball. Does design strategy or programming knowledge determine shift of focus in expert Prolog programming? In *Empirical Studies of Programmers, fifth workshop*, pages 162–186, Norwood, New Jersey, 1993. Ablex.

[14] N. Pennington. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology*, 19:295–341, 1987.

[15] R. S. Rist. Schema creation in programming. *Cognitive Science*, 13:389–414, 1989.

[16] P. Romero. Focal structures and information types in Prolog. *International Journal of Human Computer Studies*, 54:211–236, 2001.

[17] S. Wiedenbeck. Beacons in computer program comprehension. *International Journal of Man-Machine Studies*, 25:697–709, 1986.