PROGRAMMING PLANS, IMAGERY, AND VISUAL PROGRAMMING

T. R. G. Green [1] *& R. Navarro* [2]

[1] MRC Applied Psychology Unit
15 Chaucer Road, Cambridge CB2 2EF, UK
fax: +44-(0)-223-359062

[2] Departamento de Psicología Experimental y Fisiolog. Comp.
Campo de la Cartuja, Universidad de Granada
Granada, Spain
e-mail: `rnavarro@ugr.es`

KEY WORDS: visual programming, spreadsheets, Basic, mental models, mental imagery, programming plans, schema theory, Pathfinder, program comprehension

ABSTRACT: Spreadsheets and visual programming languages raise a challenge for existing schema-based models of programming knowledge, which have been scarcely been applied outside Pascal-like languages. Recent demonstrations of the role of mental imagery in spreadsheet programming raise another challenge to schema-based theories, which are propositional in form. We show that a recent schema-based model can be applied to visual languages and report comparisons between elicited mental structures for visual, spreadsheet and textual languages. Although visualness affected elicited structure (which is not predicted by schema theory), results suggest modification of schema theory rather than refutation. Programming environments should support 2D layout better.

INTRODUCTION

Understanding programmers' mental models is an important goal for cognitive science and for HCI. How do people design, build and maintain such vastly intricate mechanisms as a full-size program? How can we improve their tools? How can we improve the teaching of programming? As programming languages proliferate, especially for end-users, we also need to ask, with increasing urgency, how should languages be designed for best results?

Two streams of research have investigated mental models of programming, each with its own methodologies. One stream has investigated conceptions of the computer (or the programming language) *as a whole mechanism*. The methodology adopted has necessarily concentrated on subjects' perceptions of programming concepts, rather than the components of a particular program. Thus, programming experts organize programming keywords according to their meaning, whereas novices focus on surface features (McKeithen et al., 1981); experienced programmers form more coherent networks of concepts than intermediates or novices (Cooke and Schvaneveldt, 1988); better feedback improves the concept networks formed by novices (Cañas et al., 1994).

The other stream has investigated mental models of *particular programs*, usually conceiving them as compositions of 'plans' or 'schemas' such as a Running Total plan:

```
  Total := 0;
  {start loop}
  Total := Total + X
```

This tradition goes back to Soloway & Ehrlich (1984). The most thorough and recent statement of it is by Rist (1994), and it is his analysis that we shall use below. Appropriate methodologies have focused either on the inferences that can be drawn from particular components of the program (e.g. Détienne, 1988) or else on the perceived relationship between components (e.g. Pennington, 1987).

However, all the studies cited, and virtually all those in the literature, have used 'traditional' languages, usually Pascal, occasionally Fortran, C, or Cobol. These are all text-based languages whose underlying paradigm is procedural and imperative. There are many alternatives including logic languages, object-oriented languages, dataflow languages, forms-based languages, and spreadsheets, some of which exist in visual form as well as textual, yet very few studies have reported comparisons across paradigms. In fact, it has not been easy to see how the methods developed for text-based programs could be applied to other paradigms. Consider, for instance, McKeithen et al.'s study of keyword classification. What equivalent could be found for a visual dataflow language, which is effectively devoid of keywords? Or consider Détienne's study, in which a program was exposed line by line to the subject: what would be the equivalent in a visual programming language, which has no individual lines of code?

These difficulties suggest that schema-based theories of programming have limited applicability. Such theories would be worth little. One goal of this paper is to show how theory and experimentation can be applied to other paradigms, as is urgently needed.

Our other goal is to see whether schema theory is not only applicable, but also correct. Are programming schemas the same for all paradigms, or is anything different about the mental representation of visual programs? Saariluoma and Sajaniemi (in press) have challenged the entire position of programming theory to date, charging it (and indeed the whole of HCI) with paying too little attention to image-based mental representations and concentrating instead on theories of propositional representation. They showed that spreadsheet users make use of visual imagery in planning manipulations, a result which of course is perfectly consistent with the well-known importance of visual imagery in other kinds of problem solving. The implication is that mental images of program layout are a 'preferred resource', one that programmers would use if they could. Perhaps, therefore, the schema-based theories only describe a type of behaviour that occurs in impoverished situations, an 'unpreferred resource', and will need to be extended to deal with other paradigms.

Plans of programs

Early versions of the theory of programming 'plans' or 'schemas' had little to say about the cognitive architecture (Soloway and Ehrlich, 1984; Détienne, 1988). Rist (1994) has greatly extended the theory and developed it into a working AI program, showing how it can describe many strategies of program design and comprehension. He posits a hierarchical structure of cues and nodes. Cues are indexes into memory (knowledge of programming, understanding of a particular program), and they can specify the role (e.g. data stream), the goal (compute a total), the object (e.g. carpet rolls), or both goal and object (total of carpet rolls). In the last form they correspond to programming plans. The decomposition of knowledge into cues is accompanied by a decomposition of object structures (house = rooms = wall + floor, floor = length + width).

Cues are combined into nodes, to which they may supply data or control, from which they may receive data or control. Concrete nodes are built into code, although one line of code may combine several nodes. Nodes are hierarchically structured.

The novelty of the theory lies in its development of abstraction, allowing the same abstract component to be identified with different concrete realisations. It is this, the representational part of the theory, that is tested in the present paper. Other parts of the theory deal with the cognitive processes of memory search, yielding predictions of the relationship between expertise and the processes of program development and program comprehension; these will not be considered here.

Paradigms of textual, spreadsheet, and visual programming

Textual-imperative programs in the style of Basic will be familiar to all readers. The spreadsheet is fundamentally different because its formulae do not specify a sequence of computations, but instead specify sources of data for computations. Visual dataflow programs are similar to spreadsheets in that respect, but instead of referring to data sources by addresses on a grid, a direct line is drawn between operators to indicate the flow of data.

Programmers have considerable freedom to use two-dimensional layout in functionally meaningful ways in a spreadsheet, less in LabVIEW, and still less in Basic. Nardi (1993) has convincingly argued that the 'visual formalism' of spreadsheet layout is fundamental to their wide acceptance.

METHODS OF INVESTIGATION

Our approach draws on both the 'whole program' and the 'individual program' research traditions mentioned in the Introduction. Instead of asking how programmers perceive keywords, which have no particular status in schema-based theories, we have asked how subjects perceive the program fragments identified by schema theory. Thus we have united the two traditions for the first time.

Many techniques have been used to elicit and analyse mental representations of programs and program components: priming (Pennington, 1987), card-sorts (Davies et al., in press), multi-dimensional scaling (Cañas et al., 1994), etc. We followed Cooke and Schvaneveldt (1988) in using the 'Pathfinder' technique to analyse paired comparison ratings, because this technique yields networks that can be directly compared to the theoretical structure; unlike hierarchical analysis it does not constrain the structure of the network, and unlike multi-dimensional scaling it shows direct relationships between items.

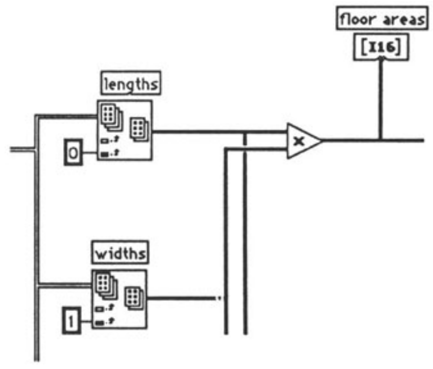
Equivalent programs in three different paradigms were analysed into theoretically equivalent fragments. After exploring the program, subjects rated pairs of fragments for similarity and for closeness of relationship. The principal question of interest lay in the structure derived from these ratings. If schema theory is correct and complete, and if the ratings are eliciting the mental representations, the structures should be identical.

EXPERIMENT

Design

Three programming languages were employed: a typical spreadsheet (we used Microsoft Excel version 4.0); a version of Basic, chosen as a 'vanilla' text language; and LabVIEW, a visual dataflow language. Two programs were written in each language. One program, used for practice, computed the total time taken for a journey given a table of distances and speeds for each leg of the journey. The second program, used for the experiment proper, took as data the dimensions of a set of rooms, plus constants for price and size of carpet rolls and of wallpaper rolls, and computed the total cost of carpets and of wallpaper (no allowance being made for doors windows etc.). Three different qualities of carpet were allowed for.

Each program was analysed according to Rist's theory. Since their computations were identical, the abstract structure revealed by the analysis was identical across languages. The concrete structures isolated by the analysis were, of course, very different (Figure 1). Note that we used the formula view for the spreadsheet, which is not the default view but which obviously is much nearer to being informationally equivalent to the LabVIEW and Basic representations than the values view.



```
FOR room = 1 to 3
  READ width (room), length (room), height(room)
NEXT room
...........
FOR room = 1 to 3
  floor_area (room) = width (room)*length (room)
  ..........
NEXT room
```

	A	B	C
1			
2		Lengths	Floor area
3		15	=B8*B3
4		12	=B9*B4
5		12	=B10*B5
6			
7		Widths	
8		13	
9		14	
10		10	

Figure 1: concrete realisation of an equivalent fragment in each language. Ellipses in the Basic indicate physical separation; all fragments were displayed in their exact physical position within the whole program.

The subject viewed a Macintosh displaying a blank window on which were located button icons. Pressing a button revealed one of the fragments. Releasing the button hid it again. For the LabVIEW and spreadsheet programs the buttons were placed roughly in the centre of the fragments they invoked; for the Basic program, the buttons were placed on the line that was identified as the 'focal line' in Rist's theory. Fragments were displayed in their exact position in the whole running program, so the effect was rather like viewing a jigsaw piece by piece until one understood the picture. (The nature of Basic means that some fragments contain physically separate statements, such as in a Running Total.) This phase of the study is similar to 'line-at-a-time' text exploration (Détienne, 1988; Robertson et al., 1990) or 'cell-at-a-time' spreadsheet exploration (Saariluoma and Sajaniemi, 1989, 1991), but with a larger grain size.

Subjects were volunteers (5 per group) from the staff and doctoral students of the MRC Applied Psychology Unit. The populations of available users of the three paradigms differed in length of programming experience and in the typical domain of programming, and therefore there is an unavoidable confounding; however, as they were all persons of very high educational achievement, there would be little point in bothering with any kind of aptitude test.

Procedure

The first phase of the experiment required subjects to familiarise themselves with a program by viewing one fragment at a time, in freely chosen order. Each subject performed the exploration task first with the practice program and then with the experimental program. They were instructed to continue exploring until they felt confident that they understood the program and could explain it to the experimenter. During their explorations, all their button pushes were recorded.

When subjects felt they understood the experimental program they told the experimenter, who asked two simple questions about its operation to verify their understanding. Subjects then started the second phase of the experiment, during which they were asked to make comparisons between pairs of fragments. The comparisons were presented on-line and ratings were indicated by pressing one of 9 radio buttons, going from 'not at all' to 'very'. For each pair of fragments two questions were asked: (1) How similar are these two fragments?, and (2) In the program you have studied how close is the relationship between these two fragments? (Question 1 is not theoretically interesting, but was included because its presence helped subjects to understand that in question 2 they should not interpret 'closeness' to mean surface similarity.)

RESULTS

Exploration Phase

During the exploration of programs all button presses and times were recorded. (Exploration data for one subject were lost because of technical problems). Individual differences were large and analysis of variance on raw and log-transformed data revealed no significant group difference in the number of button presses, the total time, the average duration of button presses, or the proportion of time spent looking at different fragments.

Comparisons Phase

The principal question of interest was whether differences existed in the mental representations elicited by the comparison questions. The Pathfinder technique can form networks based on individuals' ratings or on averaged ratings; evidently, networks averaged over groups are desirable, but precautions should be taken when using any such technique, to avoid imputing structure to data that do not in fact contain any such structure (Pathfinder, like cluster analysis, can be applied to random numbers, but the results would be meaningless, of course). The first precaution is to examine the *coherence* of the data, that is, whether a directly-elicited similarity between two entities agrees with a similarity computed indirectly from ratings of other entities.

For question 1 the coherence was rather low (mean = 0.45), suggesting that our subjects interpreted similarity in different ways for different comparisons, which was confirmed by post-experimental verbal reports. Results from question 1 were discarded. Fortunately, this was the less interesting question, and the coherence for question 2 was very much higher, varying from 0.76 (LabVIEW group) to 0.94 (spreadsheet group), indicating that subjects applied a relatively uniform criterion.

The second precaution was to correlate the similarities obtained from each subject with the mean similarities of their groups, to be sure that the means were representative. In almost all cases the correlations were very high (Spearman's rho = 0.74 to 0.94), with 3 subjects – one in each group – giving lower correlations. The network analyses described below were computed both with, and without, these 3 subjects, giving results that were minimally different. We have reported the analyses that used all the subjects.

Having taken these precautions the Pathfinder networks were computed for the question 2 ratings (Figure 2). The networks were appreciably different, and no pair of networks correlated at better than 0.2, showing that the cognitive structures were different for the three groups. In particular, the spreadsheet group produced a network which was completely separated into two parallel subnetworks, one for the carpeting and one for the papering, while the other networks intermingled these computations. (Wallpaper-related fragments have thick borders in Figure 2.) All three groups showed some direct influence of the physical layout of the program code, especially the spreadsheet group, but that was clearly not the only factor.

The first interpretation is that schema-theory is thereby refuted, since it predicts no differences, but closer examination suggests the following interpretation. Rist's schemas, it will be remembered, combine a decomposition of the goal structure with a decomposition of the object structure. The spreadsheet group produced a network which almost exactly matched the *object* structure. In contrast, the Basic group produced a network which closely matched the *goal* structure: that is, fragments that achieve similar goals are closely linked in the network, irrespective of the object (carpet or wallpaper). The LabVIEW group is somewhere in the middle, possibly combining both aspects.

Why should subjects do that? Because our spreadsheet program, following typical real-life practice (Hendry and Green, 1994), was organized by function. Thus, subjects could make extensive use of spatial position imagery as a coding. LabVIEW has less scope for that, and Basic has virtually none.

Caveat. No one experiment can settle all issues. There are some cautions needed, as follows. First, it has already been mentioned that our subjects came from different populations. Although we ourselves do not believe that the effects were due solely to population differences, the possibility cannot be discounted.

Second, there are many parochial differences between the products we compared. Excel, like most spreadsheets, is obstinately old-fashioned, insisting on using uppercase text with minimal use of whitespace to improve legibility (a sad case where the original design has persisted in the face of both common-sense and well-researched studies on legibility). The result is formulae looking like this:

```
=CEILING(A31/$E$6,1)
=E19*INDEX(D6:D8,A6)
```

The other two languages have better legibility. Again, we do not believe that these differences affected our results in any serious way.

CONCLUSIONS

First, we believe this is the first occasion that schema-based theories of programming knowledge have been extended to visual and textual dataflow languages.

Second, we have also shown that cognitive relationships between theoretically-equivalent structures are different. However, we do *not* think that these differences refute the claims of schema theories. Our interpretation is that they show that different aspects of the schema are emphasized in different situations. Differences between the textual and visual languages show a gradation from Basic to spreadsheet, closely following the degree to which the program layout reflects the functional organization.

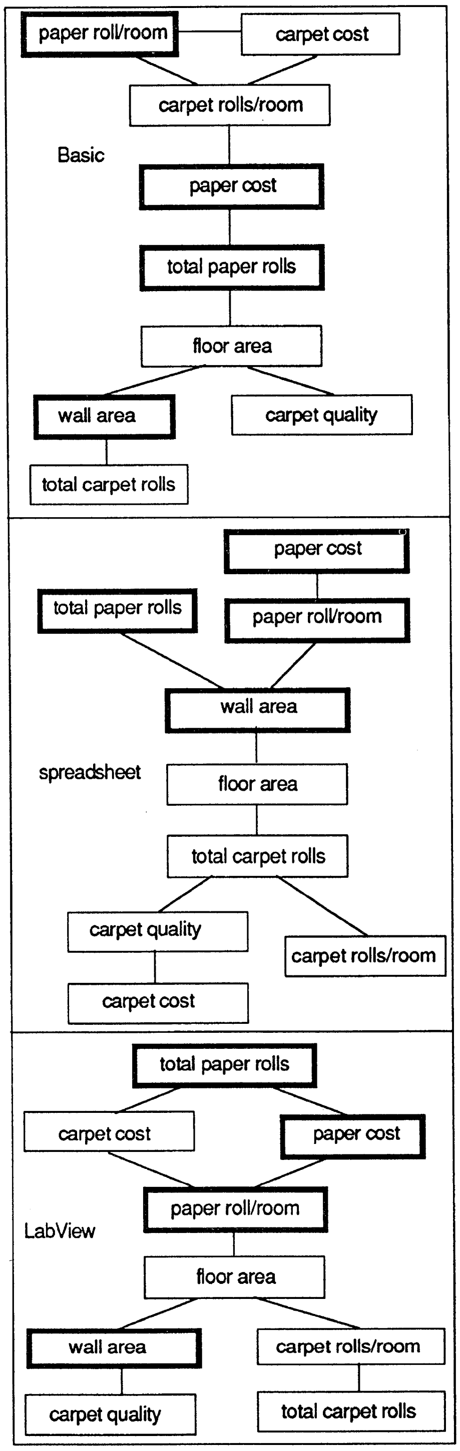


Figure 2: Pathfinder networks

The emphasis on layout apparently reconciles schema theory with the views of Saariluoma and Sajaniemi (in press). Our subjects behaved as though they did, indeed, use spatial imagery, just as theirs did, but they apparently used it not as a thing apart but to support the perception of schema-like components.

Rist's schema model could readily be modified. The cognitive processes it posits already include several ways to index nodes in memory and in the world. One of these is linear search, designed - of course - for searching through conventional textual code. The theory can be readily extended by adding location cues to its repertoire, usable only when the programming language allows location to be closely aligned with function. These findings reinforce the views expressed by Saariluoma and Sajaniemi that more work is required on integrating imagery into HCI theory.

The use of layout in this way, as a meaningful supplement outside the 'official' notation, has been termed 'secondary notation' by Petre and Green (1992). A close parallel can be found in recent developments on information visualisation, and in particular the concept of 'cost-of-knowledge' (Card et al., 1994); the inference we draw from these results is that the use of layout is preferred because the cost-of-knowledge is less, which would strongly confirm the interview-based conclusions drawn by Nardi (1993) and Hendry and Green (1994). From a development perspective, therefore, increased attention should be given to programming environments in which spatial layout can be correlated with function.

REFERENCES

Cañas, J. J., Bajo, M. T. and Gonzalvo, P. (1994) Mental models and computer programming. *Int. J. Human Computer Studies* 40, 795-811.

Card, S. K., Pirolli, P. and Mackinlay, J. D. (1994) The cost-of-knowledge characteristic function: display evaluation for direct-walk dynamic information visualizations. *Proc. CHI '94.* ACM Press.

Cooke, N. J. and Schvaneveldt, R. W. (1988) Effects of computer programming experience on network representations of abstract programming concepts. *Int. J. Man-Machine Studies*, 29 (4), 407-427.

Détienne, F. (1988) Une application de la théorie des schémas à la compréhension de programmes. *Le Travail Humain*, 51, 335-350.

Hendry , D. G. and Green, T. R. G. (1994) Creating, comprehending, and explaining spreadsheets: a cognitive interpretation of what discretionary users think of the spreadsheet model. *Int. J. Human-Computer Studies*, 40(6), 1033-1065.

McKeithen, K. B., Reitman, J. S., Rueter, H. H. and Hirtle, S. C. (1981) Knowledge organization and skill differences in computer programmers. *Cognitive Psychology*, 13, 307-325.

Nardi, B. (1993) *A Small Matter of Programming: Perspectives on End-User Computing.* MIT Press.

Pennington, N. (1987) Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology*, 19, 295-341.

Petre, M. and Green, T. R. G. (1992) Requirements of graphical notations for professional users: electronics CAD systems as a case study. *Le Travail Humain*, 55(1), 47-70

Rist, R. S. (1986) Plans in programming: definition, demonstration, and development. In E. Soloway and S. Iyengar (Eds.), *Empirical studies of programmers.* Norwood, NJ: Ablex.

Rist, R. S. (1994) Program structure and design. Unpublished report 94.4, School of Computing Sciences, Sydney University of Technology.

Robertson, S. P., Davis, E. F., Okabe, K. and Fitz-Randolf, D. (1990) Program comprehension beyond the line. In D. Diaper, D. Gilmore, G. Cockton and B. Shackel (Eds.) *Human-Computer Interaction - INTERACT '90.* Elsevier.

Saariluoma, P. and Sajaniemi, J. (1989) Visual information chunking in spreadsheet calculation. *Int. J. Man-Machine Studies*, 30, 475-488.

Saariluoma, P. and Sajaniemi, J. (1991) Extracting implicit tree structures in spreadsheet calculation. *Ergonomics*, 34 (8) 1027-1046.

Saariluoma, P. and Sajaniemi, J. (in press) Transforming verbal descriptions into mathematical formulas in spreadsheet calculation. *Int. J. Human Computer Studies.*

Soloway, E. and Ehrlich, K. (1984) Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*, SE-10, 595-609.

ACKNOWLEDGEMENTS

We thank Robert Rist, who helpfully checked our analysis; Pertti Saariluoma and Jorma Sajaniemi, for details of their method of analysis, and Jose Cañas for help with Pathfinder. Remaining errors are, of course, our responsibility. Our subjects and support staff at the APU also deserve grateful thanks. Excel and LabVIEW are trademarks of Microsoft and National Instruments Corp. respectively. This work was performed while the second author was a visitor at the MRC APU, and she thanks the director and staff for their hospitality.