Some Evidence for Graphical Readership, Paradigm Preference, and the Match-Mismatch Conjecture in Graphical Programs

Jarinee Chattratichart and Jasna Kuljis
The VIVID Centre
Department of Information Systems and Computing
Brunel University, United Kingdom
{Jarinee.Chattratichart, Jasna.Kuljis}@brunel.ac.uk

Keywords: POP-I.B. pre-programming knowledge POP-II.A. novice programmers POP-III.C. visual languages

Abstract

Research into the psychology of programming has been mostly related to textual programs. The application of the theory of programming to visual programs could be further supported by empirical evidence. Graphical readership, a skill that cannot be ignored in visual programming, has been little explored; similarly the case with regard to the effect of paradigm shift on novice programmers. Our research addresses some of these issues. This paper presents empirical evidence drawn from our experiments of first year students interacting with graphical programs. Results of these experiments provide support for the match-mismatch hypothesis, a control flow bias among novice programmers, and the possibility that prior experience with construction toys such as Lego is one of the determinants for graphical readership.

Introduction

The psychology of programming field has provided us with knowledge that helps us understand the nature of the programming process, the programmers, and the programs. However, most research in this field and the knowledge derived from it is based on empirical studies of programmers performing programming tasks on textual programs. Programming languages have evolved through changes in both paradigms and modality, so there is a need for more evidence in these new paradigms and modality to support the previous findings. In visual programming, a program is represented by a combination of graphical objects and text. Searching for information in a program involves more graphical than textual reading, and relies on perceptual cues, navigation, browsing, and interactions with the computer. The applicability of the theory of programming can be strengthened if, for example, the match-mismatch hypothesis (Gilmore and Green, 1984) still uphold in visual programs. A paradigm shift in the new languages should also cause some concerns regarding novice programmers. Although a particular programming paradigm does not affect the way experts program (Petre, 1996), there is evidence that it has some effects on them and more strongly on novices regarding program comprehension (Corritore and Wiedenbeck, 1999; Good, 1999; Wiedenbeck and Ramalingam, 1999; and Wiedenbeck et al., 1999). This leads to a question of paradigm preference: whether novices have a paradigm preference and hence would be able to cope better with learning programming in that paradigm. Another area that has not been well explored in the psychology of programming research is graphical readership skills, which can have an effect on performance in visual programming. These issues are addressed in this paper together with empirical evidence for them that emerged from our experiments on the comprehension of novices with regard to graphical programs. The evidence presented is as follows:

- The presence of the match-mismatch phenomenon in control flow graphical programs.
- Control flow bias among first year students.
- Prior experience with Lego contributes to graphical readership skills.

The paper first explores previous research relating to the evidence above, followed by a brief description of the experiments. Results are then discussed and, finally, conclusions are drawn.

Research Background

The match-mismatch hypothesis

According to Sime et al. (1977), there are two processes to extract information from a program: extracting sequence and extracting taxon information. The implication of reading and comprehending a program is that the first process (sequence) implies forward tracing and the second (taxon) implies backward tracing. In textual programs the authors found that, for sequence information, nesting is better than 'goto' style. Empirical studies have shown that the comprehensibility in textual programs very much depends on backward tracing and that nested conditionals highlight sequential information (forward tracing) (Sime et al., 1977 and Green, 1977). The match-mismatch hypothesis, derived from these studies, states that performance is best when there is a match between representation and the information required (Gilmore and Green, 1984). Therefore, performance in forward tracing should be better than backward tracing for a sequential program, and the vice versa for a circumstantial program. However, the former has not always been supported in graphical programs.

In the study by Moher et al. (1993), the Nested Petri net program, which was designed to represent a sequential program, exhibited much faster backward performance than forward performance. In fact, in all the Petri net programs used in the experiment, backward tracing outperformed forward tracing. Whitley (2000) questioned whether they had failed "*to devise a forward net representation*". Interestingly, the expectation that forward performance should be superior to the backward one had already been challenged in the studies by Curtis et al. (1989). The diagrams used in their study are like flow diagrams and the programs are hence sequential. Table 1 tabulates the mean time taken per question as approximated from the plots in Curtis et al. (1989; Figures 4 and 5, p 183-184) for diagrams that used ideogram as graphical primitives and the three spatial arrangements tested. It shows that forward tracing is slightly faster than backward tracing for the Sequential diagram only, where the term 'Sequential' relates to a specific spatial arrangement used. It appears that there is no statistical difference between the two tasks in the graphical programs used by Curtis et al. (1989).

Ideogram +	Mean seconds per question (approx.	
	Forward	Backward
Sequential	38	43
Branching	36	35
Hierarchic	39	36

Table 1 – Forward backward performance (Curtis et al., 1989).

Green et al. (1991) tested the match-mismatch hypothesis using the Boxes and the Gates notations of LabVIEW to represent sequential and circumstantial programs, respectively. The match-mismatch phenomenon was observed in this experiment. Whitley (2000) commented that the studies by Green et al (1991) and by Moher et al. (1993) differed in "*the use of visual shapes (syntax) and in the semantics attributed to those shapes*" and that the Petri net programs differed only in secondary notation, that is, in the arrangement of graphical primitives. The diagrams used by Curtis et al. (1989) also differed in secondary notation only. Could it be that the results from studies by Moher et al. (1993) and by Curtis et al. (1989) showed evidence of the effect of secondary notation overriding the match-mismatch effect? This issue will be discussed in the following sections.

Programming paradigm

There is evidence that programming paradigms affect the program comprehension of novices. Some within-study research show that the mental representation of programs held by novices is program-oriented for procedural languages (Pascal, C, and a control flow VPL) and is domain-oriented for an object-oriented language (C++) and a data flow VPL (Good, 1999; Wiedenbeck and Ramalingam, 1999; Wiedenbeck et al., 1999). Furthermore, procedural languages seem to be easier for novices. In an experiment comparing the ability to make queries in SQL (a nonprocedural query language) with TABLET (a procedural query language), Welty and Stemple (1981) found that the TABLET performance was better than the SQL performance for difficult queries. In long programs written in C++ and in Pascal, both novices using C++ and novices using Pascal exhibited a program model mental representation (Wiedenbeck et al, 1999). In addition, overall performance for the Pascal group was 15% better. Good's experimental results also show "*control flow supremacy*" (Good, 1999). The overall performance of novices for the control flow VPL was higher than for the data flow VPL. The other interesting observation by Good was that students might be already control flow biased because their previous programming experience was dominantly procedural. Davies (2000) compared short and long viewing comprehension performance between experts and novices across all information types. His data for the novice group indicate that control flow performance is strongest among all information types in both short and long viewing tests.

In summary it appears that programming paradigms affect accessibility of information differentially, thereby affecting the comprehension ability of novices with different information types in programs. Novices seem to find control flow easier than both data flow and object-oriented programming. And, finally, student programmers may be control flow biased by the time they begin their studies in universities.

Graphical readership skills

Programmers' performance depends on how good they are in observing and interpreting the meaning of perceptual cues available in visual programs. Petre and Green (1993) maintain that these skills have to be learned and acquired through experience. It therefore follows that graphical readership skills of VPL novices will improve with training. However, some measures of these skills can be useful in predicting programmer performance, designing training materials, and designing experiments. So far, researchers have not been successful in finding a correlation between some cognitive tests such as paperfolding and pathfinding and (graphical) programming performance (Good, 1999). This paper presents evidence for prior experience that could contribute to graphical readership skills.

Experiments

The evidence presented here is gathered from experimental and questionnaire data. The experiments generally looked at the effect of diagram layout style and directional representation on the graphical program comprehension of first year students. In this paper we detail here the relevant statistics from our findings.

Descriptions of layout styles compared

The layout styles we used differ in the way that the diagram is to be traversed and how its graphical primitives are placed. Our graphical programs are boxes-and-wires diagrams, each made up of a unique layout style. Six layout styles were compared in two experiments. Not all styles were used in each of the two experiments. The layout styles are:

- Top Down (TD)
- Hierarchical-Nested (HN)
- Bowles (BS)
- Linearised Net (LN)
- Rectangular Net (RN)
- Curvy Net (CN)

The control flow and data flow representations for the above are given in Figure 1 (a and b) and in Figure 2, respectively.

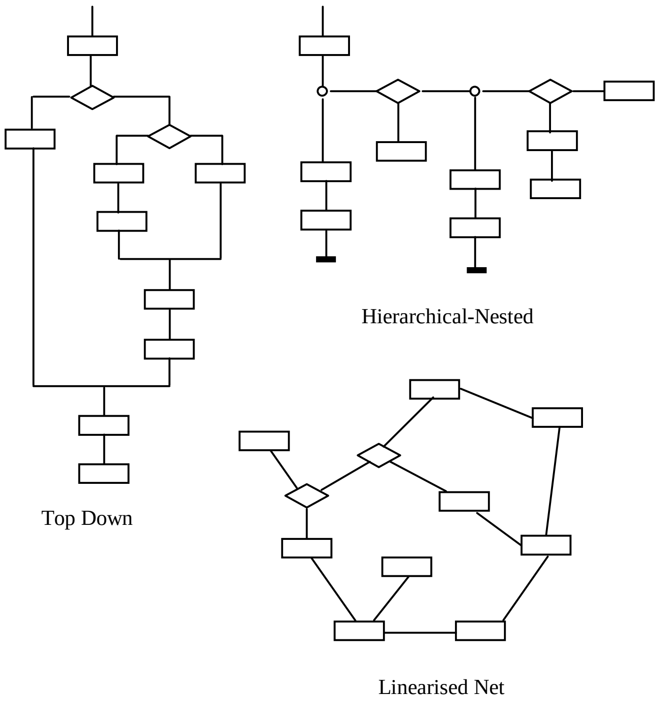
Experimental procedures

Experiment I.

This experiment compared the comprehension of graphical programs with a conventional textual program (see Figure 4) in the control flow and the data flow paradigms. The graphical programs differed in layout style: TD, HN, and LN. An arrow was used as the directional representation. The experiment was a within-subjects design. Participants answered four forward questions and four backward questions, one pair for each program. All participants took part in both the control flow experiment and the data flow experiment. Half of the participants did the control flow experiment first while the other half did the data flow experiment first. Participants saw four program representations: TD, HN, LN, and Text, in each experiment.

Experiment II.

This experiment compared comprehension in control flow graphical programs. These programs differ in layout style and directional representation. The layout styles used are TD, HN, BS, RN, CN and the two directional representations: arrow and line. The experiment was a 5x(2) mixed factorial design. The between-subjects factor was layout style. The within-subjects factor was directional representation. Participants were divided into five groups. Each group was presented with one layout style, each with two directional representations: arrow and line. Each participant answered four



forward questions and four backward questions for each of the two programs; 16 in total. The equivalent textual program is given in Figure 5 for reference only.

Data from twenty-one participants in experiment I and sixty participants in experiment II (twelve per group) were analysed. In both experiments the order of programs, question, and question type was randomised to counter balance order effects.

Figure 1a.- Control Flow Layout Styles.

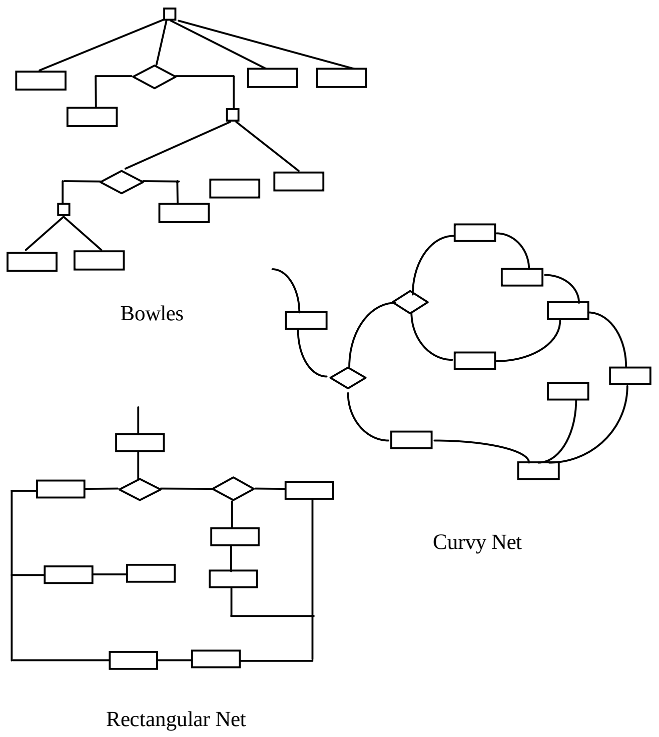


Figure 1b.- Control Flow Layout Styles.

Results

1 Forward vs backward response time performance

Experiment I

ANOVA revealed no main effect of question type for either the control flow or the data flow experiment. In the control flow experiment however, backward tracing took significantly longer than forward tracing with the textual program, t(20) = 2.117, p = 0.047.

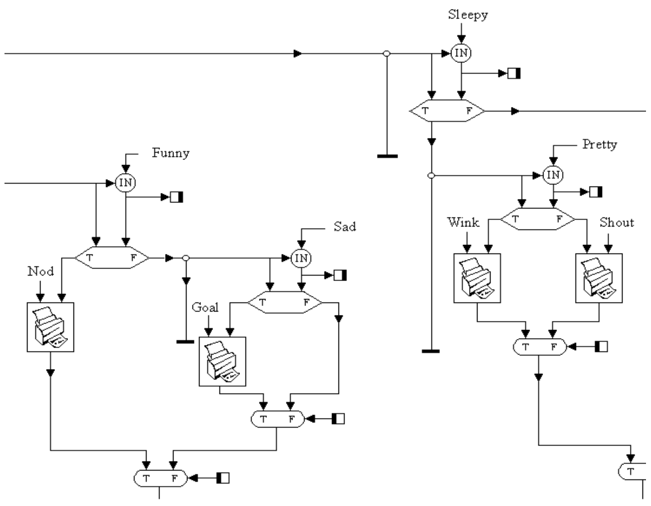


Figure 2.- Experiment I: Snapshot of the data flow program for HN.

Experiment II

ANOVA was performed separately for the arrow and the line programs. They revealed the main effects of question type:

- For arrow, F(1,55) = 59.763, $p < 0.001$.
- For line, F(1, 55) = 34.777, $p < 0.001$.

There was no interaction, question type x layout style, for either the arrow or the line. Pairwise comparisons between forward and backward performance showed that backward tracing took significantly longer than forward tracing in all layout styles and in both arrow and line representations. All t-tests were significant at $p < 0.01$ for arrow and at $p < 0.05$ for line. Figure 6 plots the performance data for the arrow and the line programs separately.

2 Paradigm preference

Experiment I

A significant paradigm effect on response time performance was found. However, there was no main effect on accuracy performance.

Participant data were then split into two groups, according to the programming paradigm of the experiment they took part in first. The first group (*control flow*) consisted of participants who were trained in the control flow paradigm and who did the control flow experiment before the data flow. The second group (*data flow*) consisted of those who were trained in the data flow paradigm and who did the data flow experiment before the control flow one. Figure 3 shows that the *control flow* group performed equally well in both the control flow and the data flow programs. However, the *data flow* group performed more poorly in the data flow programs than in the control flow programs.

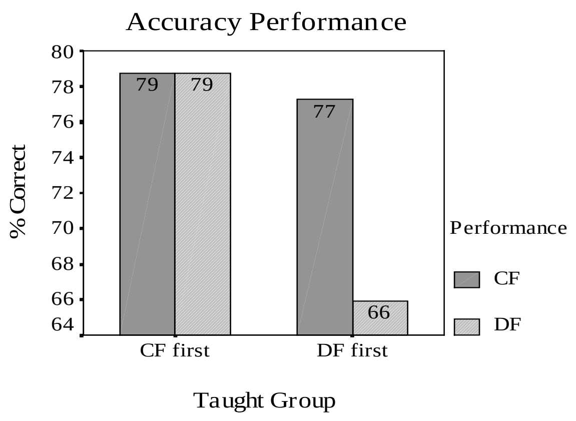


Figure 3.- Control flow and data flow accuracy performance of the two taught groups.

Questionnaire data

In the questionnaires given to participants and would-be participants during the course of our experiments we asked about their previous programming language experience. The respondents were first year students doing computer studies at three different universities in the UK. The questionnaires were collected on four different occasions between November 1999 and November 2000. Of the 131 respondents, the languages previously known are 74.1%, 21.0%, and 4.9% for procedural, object-oriented, and declarative languages, respectively. The language that they were learning at the time was not included in the calculation.

3 Graphical readership skills

Cognitive test

Because the design of experiment II was a mixed factorial, would-be participants were given a cognitive test as a pre-test so that the test results could be used to assign participants to groups. The

Choosing a Path Test (pathfinding), taken from the Kit of Factor-Referenced Cognitive Tests (Ekstrom et al., 1976) was used. Due to the time constraint, only the first test, which consisted of sixteen questions, was given and time was limited to seven minutes as required by the Kit. Seventy-nine would-be participants took the test. Only forty-one went on to participate in the experiment proper in the following week. Data from these participants show no correlation between the cognitive test result and that of the experiment proper.

Questionnaire data

Because we did not find a correlation between the cognitive test and the performance in the experiment proper, a one-page post-experiment questionnaire was designed in order to find out what prior experience could have helped making the difference in graphical readership skills. Of the sixty-three participants in experiment II, forty-two responded. A Discriminant Analysis was performed on these data. Eleven independent variables used were: previous programming experience; academic achievement; interest in board games; London underground map reading skill; experience with computer and Nintendo games; interest in DIY; interest in drawing; interest in construction toys such as Lego; having a PC at home; gender; and previous experience with flow diagrams. The dependent variable was the accuracy performance of the experiment, broken down into four levels according to which quartile the participant's performance belongs. The analysis gave only one function and one independent variable for the discriminant function with 48.8% success rate in classification. The discriminant function was the interest in construction toys such as Lego. The mean for the group with regard to interest in construction toys was 3.9 on a scale 1 to 5. The mean for each group ranges from 3.6 to 4.0.

Discussion

Match-mismatch phenomenon

All programs, both graphical and textual, used in the experiments are sequential in nature. According to the match-mismatch hypothesis, forward performance should be better than backward performance. The result from the first experiment did not support this hypothesis. This agrees with the previous research on graphical program comprehension (Curtis et al., 1989 and Moher et al. 1993). This casts doubt on the applicability of the match-mismatch hypothesis to graphical programs. Nevertheless, our results from experiment II show strong support for the hypothesis. What, then, could be an explanation for this discrepancy?

In the study by Curtis et al. (1989), diagrams were presented to participants by way of a sheet of paper. We inferred from Moher et al. (1993) that the Petri net diagrams seen by their participants occupied one screen per program. Our first experiment used a short program. Though it was deeply nested, scrolling was rarely required. In these programs, secondary notation played an important role in enhancing visibility of the graphical objects and tracing tasks. Its effect overrode that of the match-mismatch effect. The diagrams used in the second experiment were longer, and scrolling was required to a varying degree and hence the effect of secondary notation was overridden by the match-mismatch effect.

Control flow bias among student programmers

Statistical analysis of the first experiment did not reveal a significant paradigm difference in accuracy performance. However, it appeared that the *control flow* group (participants who were taught the control flow programs before the data flow programs) could do the data flow programs equally as well as the control flow programs. This was not the case with the *data flow* group whose performance in the data flow programs was much worse than that in the control flow programs. This might be an indication that the participants found control flow programs easier to learn as a first programming language than data flow programs. Or, it might indicate that the students were already control flow biased before they started university education. Indeed, our questionnaire data showed that 74.1% of previously known programming languages comprised procedural languages.

Graphical readership skills

Participants in the second experiment were all new entrants to the Brunel University. Although diagram-reading skills would depend on the experience and training (Petre and Green, 1993), it is not known what previous experience could affect the ability to read technical diagrams among novices. Results from the Discriminant Analysis showed that previous experience in programming languages and flow diagrams (the factors most likely to affect the experimental results) were not the predicting variables for diagram-reading ability. Interestingly, interest in construction toys such as Lego was found to be the best candidate for predicting the readability of diagrams in our experiment. Construction toys such as Lego are usually supplied with diagrammatic instructions. Those who are well experienced in playing with these toys must have acquired their skills through reading such instruction sheets and through manipulating the toys. However, we speculate that this is not the only possible prior experience that helps improving graphical readership skills. We would like to see further research in this area.

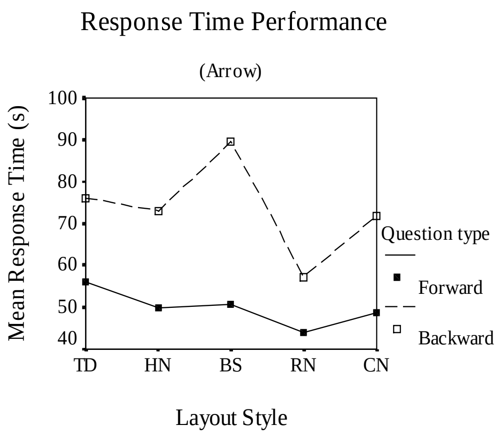
```
If S =  '*Bad*' then
 If S = '*Pretty*' then
  Loop begins for Times = 1 to 2
   If S = '*Sad*' then
    Print 'Shout'
   Else
    Print  'Goal'
   End if
  End loop
 Else
  If S = '*Funny*' then
   Print 'Nod'
  Else
   If S = '*Sad*' then
    Print 'Goal'
 End If
  End If
 End If
Else
 If S = '*Sleepy*' then
  If S = '*Pretty*' then
   Print 'Wink'
  Else
   Print 'Shout'
  End if
 Else
  Loop begins for Times = 1 to 2
   If S = '*Funny*' then
    Print 'Nod'
 Else
    Print 'Wink'
   End if
  End loop
 End if
End If
```

Figure 4.- Textual program: Experiment I.

```
  Vegetables
  If  Careful  then
   If  Sad  then
    Sausages
    Milk
   Else
    Bread
    Crisps
   End If
Fish
  Else
   Bread
   Milk
   If  Funny  then
    Cake
   Else
    Fish
   End If
  End If
  Eggs
  If  Picky  then
   Chicken
  Else
   Butter
  End If
  If  Friendly  then
   Jam
  Else
   Salt
  End If
  Pay Bill
```

Figure 5.- Textual program: Experiment II

Note that the lines are used only for the purpose of easier readability.



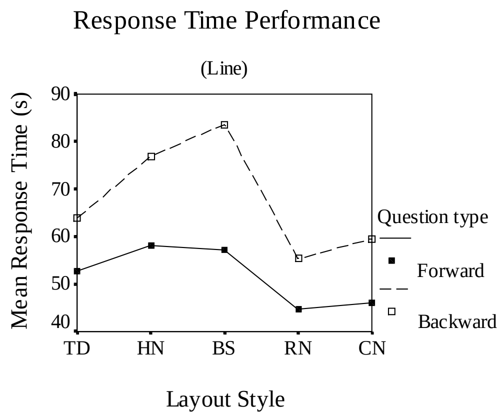


Figure 6. –Experiment II: Forward and backward performance for arrow diagrams (top) and for line diagrams (bottom).

Conclusion

We have presented evidence that confirms the applicability of the theory of programming to graphical programs. The match-mismatch hypothesis is supported by the results in our experiments. Other evidence may help in stimulating a new research direction in the psychology of (visual) programming. That student programmers are control flow biased could have an implication on the design of the programming curriculum and of new programming languages. We have also provided some evidence that prior experience with construction toys such as Lego may have made its contribution to graphical readership skills.

Acknowledgements

This research is supported by the UK Government Agency, the Engineering and Physical Sciences Research Council, grant number GR/M08387. We are grateful to all our participants and questionnaire respondents.

References

Corritore, C. and Wiedenbeck, S. (1999) Mental representations of expert procedural and object-oriented programmers in a software maintenance task, *International Journal of Human-Computer Studies*, 50, 61-83.

Curtis, B., Sheppard, S. B., Bailey, E. K., Bailey, J. & Boehm-Davis, D. A. (1989) Experimental evaluation of software documentation formats, *The Journal of Systems and Software*, 9, 167-207.

Davies, S. P. (2000) Expertise and the comprehension of object-oriented programs, *Proceedings of the 12th Annual Meeting of the Psychology of Programming Interest Group*, 61-65.

Ekstrom, R. B., French, J. W., Harman, H. H., and Dermen, D. (1976). *Manual for the Kit of Factor-Referenced Cognitive Tests.* Princeton, NJ: Educational Testing Service.

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

Good, J. (1999) *Programming Paradigms, Information Types and Graphical Respresentations: Empirical Investigations of Novice Program Comprehension*, Unpublished PhD Dissertation, Edinburgh University, UK.

Green, T. R. G. (1977) Conditional program statements and their comprehensibility to professional programmers. *Journal of Occupational Psychology*, 50, 93-109.

Green, T. R. G., Petre, M. & Bellamy, R. K. E. (1991) Comprehensibility of visual and textual programs: a test of superlativism against the 'match-mismatch' conjecture. In J. Koenemann Belliveau, T. G. Moher and S. P. Robertson (Eds.), *Empirical Studies of Programmers: Fourth Workshop*, Ablex, 121-141.

Moher, T. G., Mak, D. C., Blumenthal, B. & Leventhal, L. M. (1993) Comparing the comprehensibility of textual and graphical programs: the case of petri nets. In C. R. Cook, J. C. Scholtz, and J. C. Spohrer (Eds.), *Empirical Studies of Programmers: Fifth Workshop*, Ablex, 137-157.

Petre, M. (1996) Programming paradigms and culture: implications of expert practice. In M. Woodman (Ed.), *Programming Language Choice: Practice and Experience*, International Thomson Computer Press, 29-44.

Petre, M. & Green, T. R. G. (1993) Learning to read graphics: some evidence that 'seeing an information display is an acquired skill, *Journal of Visual Languages and Computing*, 4, 55-70.

Sime, M. E., Green, T. R. G. & Guest, D. J. (1977) Scope marking in computer conditionals-a psychological evaluation, *International Journal of Man-Machine Studies*, 9, 107-118.

Welty, C. and Stemple, D. W. (1981) Human factors comparison of a procedural and a nonprocedural query language, *ACM Transactions on Database Systems*, 6(4), 626-649.

Whitley, K. N. (2000) *Empirical Research of Visual Programming Languages: An Experiment Testing the Comprehensibility of LabVIEW*, Ph.D. Dissertation, Computer Science Department, Vanderbilt University, Nashville, TN 37235.

Wiedenbeck, S. and Ramalingam, V. (1999) Novice comprehension of small programs written in the procedural and object-oriented styles, *International Journal of Human-Computer Studies*, 51, 71-87.

Wiedenbeck, S., Ramalingam, V., Sarasamma, S. and Corritore, C. L. (1999) A comparison of the comprehension of object-oriented and procedural programs by novice programmers, *Interacting with Computers*, 11, 255-282.