

# Does Metaphor Increase Visual Language Usability?

A.F. Blackwell  
Computer Laboratory  
University of Cambridge  
Cambridge CB2 3QG, UK  
Alan.Blackwell@cl.cam.ac.uk

T.R.G. Green  
Computer Based Learning Unit  
University of Leeds  
Leeds LS2 9JT, UK  
Thomas.Green@ndirect.co.uk

## Abstract

*Previous research suggests that graphical metaphor should increase the usability of visual programming languages (VPLs) by providing an instructional aid when learning to use the language. This paper describes three experiments which manipulated the degree of metaphor in VPLs. In the first, an implicit pictorial metaphor was added to a dataflow language, while in the second and third an explicit explanatory metaphor was provided for diagrams showing computational concepts. In both cases, the performance of novices was compared to that of expert programmers, in order to judge the improvement in usability that might result from this instructional device. The resulting benefits of metaphor were smaller than expected, especially relative to the advantage of experience. This suggests that metaphor may not be an essential component in the usability of VPLs.*

## 1. Introduction

Do visual languages provide special usability benefits for novices learning how to write programs? It seems clear that they enable instructional techniques that would not be available with textual languages [3][4]. One of these opportunities for improved usability is the capability to incorporate an instructional metaphor into the visual representation. This is widely believed to be a critical factor in improving usability of other graphical user interfaces, even outside the domain of programming. For example, Microsoft's Windows Interface Guidelines for Software Design states "Familiar metaphors provide a direct and intuitive interface to user tasks. By allowing users to transfer their knowledge and experience, metaphors make it easier to predict and learn the behaviors of software-based representations." [22].

Metaphorical instruction has long been proposed as a means of making conventional programming languages easier to master. In 1975 Mayer [19] demonstrated the advantages of using an instructional metaphor when teaching the FORTRAN language. This metaphor was

essentially a cover story (involving paper work in an office) for the behaviour of the "virtual machine" constituted by the FORTRAN compiler and runtime environment. A metaphorical cover story of this type gives the learner a framework in which to understand the effects of their actions. Chee [9] has suggested that the quality of such a metaphor can be formally evaluated in terms of the quality of the analogical structure mapping from the metaphor domain to the programming language domain.

### 1.1. Metaphor and mental models

If learners are not given an explicit cover story of this type, they still construct their own mental model of the machine, as has been observed by Young with pocket calculators [44] and Tauber with operating systems [36]. Unfortunately if novice programmers are not given a model of a virtual machine, they may invent an inappropriate explanation [6][13][14], working from sources of information such as observing a debugger [7] extrapolating from tutorial code examples [24], or imagining the behavior of the machine from the viewpoint of an internal agent [40]. It seems reasonable that teachers should anticipate this by providing some explicit story of what is happening inside the virtual machine – what du Boulay, O'Shea & Monk [11] called "the black box inside the glass box".

How can visual languages help to achieve this goal? It has been suggested (e.g. by Esteban, Chatty & Palanque [15]) that the visual representation should make the virtual machine more explicit by depicting its behaviour pictorially. This might be an illustration of a metaphorical cover story like the FORTRAN one created by Mayer [19]. Kahn's ToonTalk [18], for example, presents a pictorial metaphor of cartoon characters acting in a computational town of function houses and message-carrying birds. A more conventional approach than ToonTalk is to depict the virtual machine as if it were a physical machine [16] which can be understood by analogy from the user's experience of real world machines [36][39]. Once again, the importance of this approach is supported by the fact that when an explicit metaphor is not provided, novice

programmers create their own metaphors to explain the behavior of the language [17].

## 1.2. Metaphor and paradigm

A visual programming language constructed according to these principles must serve a dual purpose: a notation for solving a problem in some problem domain, as well as the representation of some virtual machine [37][2]. Its effectiveness will depend on how well information can be mapped from the problem domain to the notation, and from the notation to the virtual machine [31][26]. The structure of the virtual machine, and the nature of the mapping, combine to form the programming paradigm – a family of languages that support the same programming techniques. Overly complex mappings can easily compromise performance [23], so one of the main challenges in improving programming language usability is providing metaphors (and hence paradigms) that are appropriate for the user [8][28][27], allowing them to learn the behaviour of the language but also to achieve useful tasks in their problem domain. Ideally, a visual programming language for novices should completely integrate the programming paradigm and the user interface metaphor.

## 2. Experiment A: Pictorial metaphor

This experiment was designed to test the efficacy of a visual programming language designed with an explicit pictorial metaphor that depicts a virtual machine. The language was a dataflow language, of the type commonly used in commercial VPLs such as LabVIEW or VEE. However it was simplified substantially, so that it could be taught to novice programmers within a two hour experimental session, including time for evaluation of learning. It included 11 basic dataflow elements (selectors, distributors, guards and accumulators), along with data sources and sinks, and data stream management.

We created two versions of the pictorial presentation of this language – one represented the dataflow paradigm using the metaphor of balls rolling through a physical machine. The syntax elements of the language were represented by cartoon-like pictures of balls rolling through mechanisms for selection, distribution etc., although these mechanisms were not intended to be realistic – subjects were told to imagine that the components worked by magic. Components were connected together by tracks showing balls rolling along them.

In the second version of the language, syntax elements were represented by purely geometric symbols, and components were connected by unadorned lines. This is more typical of commercial dataflow languages, where the metaphor is described in instructional material (LabVIEW uses the metaphor of electrical wiring), but not presented

pictorially. Figure 1 shows two versions of a selection component, in the versions with and without the rolling ball metaphor.

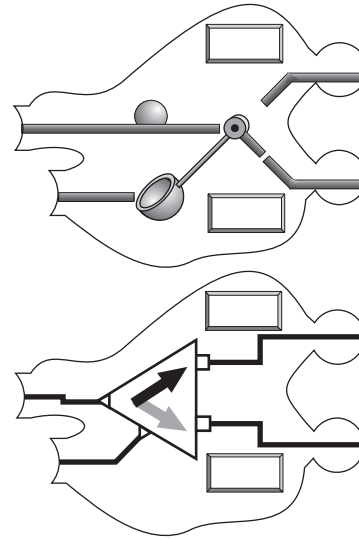


Figure 1. Selection with and without metaphor

The component nodes and connections between them were given to subjects in the form of paper cut-outs, whose profiles were identical in each version. Subjects arranged these cut-outs on a tabletop by matching the right and left hand sides of the nodes. The table top was arranged in a way that resembled typical VPL environments. Divided trays containing piles of each component were placed at the left hand side of the work surface, made to resemble a software “palette”. The behavior of each component was described on individual printed pages as if in a language manual or on-line help. These descriptions, and all instructions given by the experimenter, were identical in both experimental conditions – neither referred to the metaphor.

### 2.1. Method

Twelve experimental subjects were recruited from volunteers registered with the Subject Panel of the MRC Applied Psychology Unit in Cambridge, and from among staff at the Unit. They were divided into two groups on the basis of their prior experience of computer programming. Four subjects had previously worked as professional programmers (“*experts*”); eight had never written a computer program (“*novices*”). The novices were broadly matched in educational achievement and intelligence to the experts (educated to degree level, and IQ scores between 120 and 140). The novice group were then divided into four who were given the metaphorical version of the language, and four given the non-metaphorical version. The design compared the expert group to these two training groups. (Of the four experts, two used the metaphor version and two did not. This simply establishes

a mean for comparison, however – we did not expect the metaphor to provide training effects for experts).

The experiment involved four programming tasks, each related to the processing of bank statements, in the course of which subjects also learned all the elements the language. In the first task, the experimenter demonstrated the creation of a program to add total credits and debits, describing the operation of the program as each component was added. This description was identical in the two versions of the language. After the program was complete, the experimenter shuffled the paper components, and asked the subject to reassemble to program while thinking aloud – this verbalisation was recorded and transcribed.

In the second task, the experimenter showed a completed program to the subject, who was told that it was designed to find missing check numbers in a sequence. He then asked the subject to explain how it worked. The program contained an error (it compared each number to itself rather than to the previous one in the sequence), and the subject was asked to find this error and suggest a way of fixing it. In the third task, the subject was given a description of a program that would check the locations of cash withdrawals, and display any that had been made outside Cambridge. He or she was then asked to create a program that would carry out this function. In the final task, the subject was asked to implement a program that would find a check paid to a particular recipient, and automatically write another check for the same amount.

## 2.2. Results

This experiment tested the hypotheses that the provision of a diagrammatic metaphor will make it easier for novices to learn to write computer programs, and that this assistance will reduce the differences in performance that would normally be expected between novices and expert programmers.

Performance was assessed using two dependent variables. The first was the total *time* that the subject took to complete each of the four activities described below. The second was the degree of *elaboration* that the subject produced in the two program construction tasks. The program specification that they were given allowed several different solutions, but complete solutions had similar degrees of complexity, while incomplete solutions omitted some of the required components. The relative quality of the solutions might be subject to interpretation, but the degree of completeness could be estimated by the simple measure of the number of nodes included.

The four expert subjects showed little variation resulting from use of the metaphor. This is as expected, given that experienced programmers have knowledge about programming that is independent of the representation being used [20], and that they apply consistent strategies for code comprehension even when using a language that does not explicitly reveal their

preferred structures [5]. The main analysis therefore compared the performance of the four novices given the pictorial metaphor to that of experts, and to that of novices given no metaphor. The value of the metaphor can be determined by the extent to which their performance is raised toward the expert level. Figure 2 shows time and elaboration for the three groups.

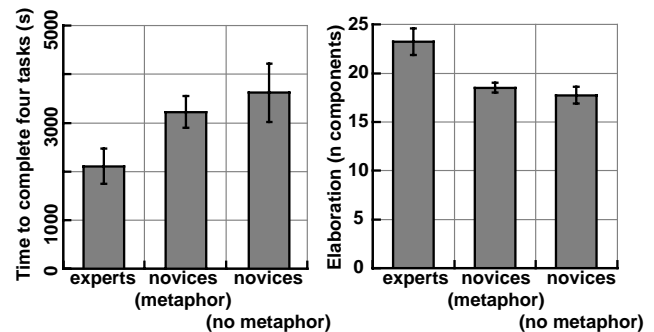


Figure 2. Implicit metaphor versus expertise

There were significant differences between novices and experts both in time to complete the task,  $F(1,10)=6.07$ ,  $p<.05$  and in elaboration,  $F(1,10)=19.66$ ,  $p<.01$ . This demonstrates that the performance measures are successful in measuring relative expertise. Nevertheless, there is no significant difference as a result of improved learning between the two novice groups, either in time or elaboration ( $F(1,6)=0.33$ ,  $p=.586$  and  $F(1,6)=0.57$ ,  $p=.477$  respectively).

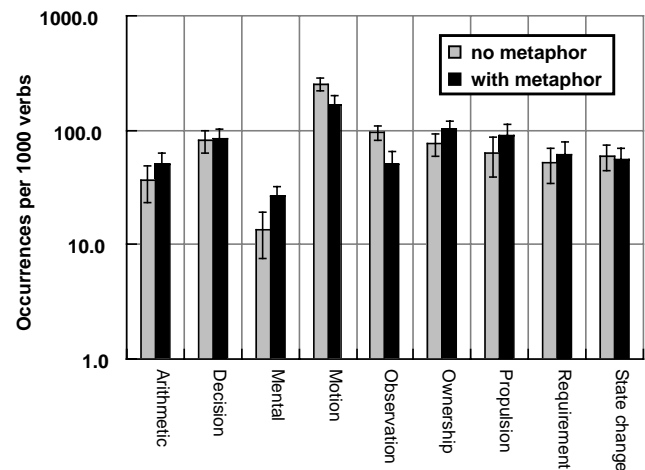


Figure 3. Verb categories in protocol transcripts

This experiment assumed that providing a pictorial metaphor would be beneficial to novices, whether or not it was explicitly explained (the same assumption that is made in many graphical user interfaces). In order to test the extent to which novices used the metaphor, we compared the vocabulary used in the verbal transcripts. Noun phrases were divided into references to the virtual machine, references to the metaphor, and references to the problem. We found that none of the participants in the metaphor

group ever referred to the implicit metaphor while thinking aloud. We also compared verb phrases, counting the verbs of motion that might be used when reasoning metaphorically about dataflow, and comparing them to other ways of describing computation. As shown in figure 3, we found no significant difference in the proportion of motion verbs used by novices with the pictorial metaphor.

### 2.3. Discussion

The measures used in this experiment were successful in measuring different levels of performance – they clearly distinguished between the expert and novice groups. Despite the fact that the measures appear to be sufficiently sensitive, the expected improvement in performance when novices were given a metaphorical diagram was not observed. Some previous experiments in HCI areas other than programming have also failed to find improvements in performance when metaphors are introduced (e.g. Eberts & Bitianda [12] or Sutcliffe & Patel [35]), but those studies have generally suggested that specific usability problems reduced the benefits expected of an otherwise valuable interaction metaphor.

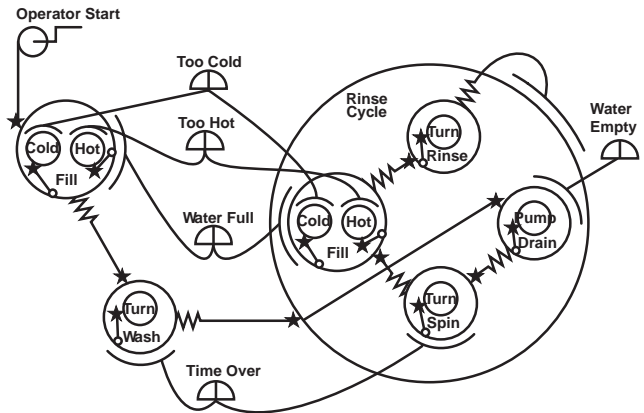
It is possible that novices in this experiment simply ignored the illustrations, as their attention was never drawn to them explicitly. This behavior has been observed in studies of illustration by Wright, Milroy and Lickorish [43]. It may even be a sensible strategy for readers whose comprehension and reading speed would otherwise be reduced by the effort involved in integrating text and illustrations, as observed by Willows [42] in studies of reading speed in children, and by Mayer and Sims [21] in a comparison of students with high and low scores on spatial reasoning tests. Some experimental subjects even made comments about the apparent complexity of the pictorial version – in informal discussions at the end of experimental sessions, subjects who were shown both alternatives often expressed a preference for the cleaner graphics of the non-metaphorical version.

The second experiment was designed to avoid the problems of graphic complexity and of subjects ignoring the pictorial metaphor by using only one visual format, but providing an explicit instructional metaphor. The complexity of the tasks was also reduced, in order to isolate the value of metaphorical diagrams for specific programming constructs.

## 3. Experiment B: Explanatory metaphor

The metaphorical interpretation in this experiment was provided in explanatory text, in a way that is more typical of current VPLs – that is, the diagrams are composed of simple geometric elements rather than pictures, and users are only aware of the metaphor if they are given explicit instruction. It also used experimental tasks that did not, on the surface, resemble computer programming. Subjects

were not told that the experiment had any relationship to computer programming. Instead, we presented diagrams using (slightly contrived) non-software task contexts to express concepts that are more often found in computer programs than in everyday life: control flow, closure, database join and visibility. An example of a control flow diagram, illustrating a washing machine cycle, is shown in figure 4.



**Figure 4. Control flow diagram (washing machine)**

The explanation of each diagram incorporated either a *systematic* metaphor that compared the graphical elements to some physical situation with an appropriate structure, or a *nonsense* metaphor that compared them to an irrelevant physical situation. The example in figure 4 described a washing machine cycle, where processes must be started, stopped or repeated according to conditions such as water level and temperature. Each process is shown as a circle, which may include other sub-processes. The signal to start a process is shown by a line with a star at the end, while a stop signal is shown by an arc at the end. One process can be made to start when another stops, shown by a jagged line at the first process.

For this diagram, the systematic metaphor described the process circles as turning cogs. The star showing when a process starts resembles a starting cog, while the arc showing when it stops resembles a brake shoe. The jagged line is a spring which rebounds when the cog that it is attached to stops. The nonsense metaphor, in contrast, described the circles as rock pools, with the stars and jagged lines resembling starfish and worms. The other diagrams were presented in similar terms, with appropriate systematic and nonsense metaphors describing their intended interpretation.

### 3.1. Method

Sixteen experimental subjects were recruited from two different populations. Eight were volunteers from the APU Subject Panel, none of whom had any experience of computer programming (“novices”). The other eight were

experienced programmers employed at the Advanced Software Centre of Hitachi Europe Limited (“*experts*”).

The experimental material was presented as a bound booklet containing both explanations of the diagrams and comprehension tests. Subjects were instructed to work through the booklet in order, without turning back to look at previous pages. At the top of each page was a box where the subject wrote the time (in seconds) when they started work on that page. Each subject was presented with all four diagrams, two of which were explained with systematic metaphors, and the other two with nonsense metaphors. The order and allocation of these conditions were balanced across all subjects.

The booklet included three tasks to be performed using each diagram: *comprehension* questions relating to an example of the diagram, *drawing* missing graphical elements on an incomplete diagram, given text describing the missing information in terms of the problem domain, and *writing* missing labels on an incomplete diagram according to the constraints in a problem domain description.

There were two dependent variables. The first was the *speed* with which subjects completed each task. The second was *accuracy* of each response. We calculated a normalized accuracy score for each task, in which the lowest score was assigned a normalized value of zero, and the highest a value of 100. All other scores were normalized by a linear interpolation between these values.

### 3.2. Results

The first hypothesis was that experts would perform better than novices. The times taken by each group to complete the three tasks, as well as the mean accuracy achieved by each group, were compared in an analysis of variance (ANOVA). This indicates that experts were significantly more accurate, with an average score of 61% whereas novices had an average score of 46%,  $F(1,14)=5.48, p<.05$ . Experts did not finish the tasks more quickly than novices – in fact they spent slightly longer, but this difference was not significant.

The second hypothesis was that a systematic metaphor would bring novice performance closer to that of the experts. Figure 5 shows the interaction between expertise and metaphor type. As expected, accuracy is poorer when the nonsense metaphor has been given. This difference does appear to be greater for novices than for experts, but the interaction is not significant. The effect of this interaction on the time taken to complete the tasks indicates that novices spend more time trying to use the nonsense metaphor, while experts spend less time. This interaction is not significant either, however.

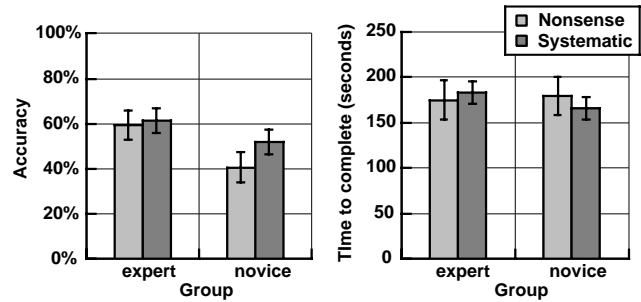


Figure 5. Explicit metaphor versus expertise

There was a significant interaction of task type with experience,  $F(2,28)=5.60, p<.01$ . Experts performed best in the task where a diagram was completed by drawing in missing elements. Novices performed more poorly on this task than on other tasks, with an average drawing score of 34%, versus 65% for experts. If the effect of metaphor is considered separately for each task type within the novice sample, metaphor appears to have had no effect at all on drawing task performance.

### 3.3. Discussion

As in experiment 1, the performance measures used in this experiment do provide a distinction between novice and expert performance which sets a reference point for the amount of benefit that metaphorical instruction provides to novices. Unlike experiment A, experts did not perform these tasks significantly faster than novices. They did achieve greater accuracy, however. In experiment A a metaphor had little further effect on the performance of experts, and that finding was repeated here. This is not of great concern, as this study concentrates on the benefit that metaphor can provide for novices. However, as with the implicit metaphor of experiment 1, it is surprising that the explicit explanatory metaphors in this experiment provided little benefit for novices.

## 4. Experiment C: Bad metaphor or none?

The results in the first two experiments might be criticised on the grounds that the metaphors were simply poorly designed. Even in experiment B, where we created an intentionally poor metaphor, it is possible that the two metaphors could be roughly equivalent in the amount of instructional benefit that they provided (for example, the nonsense metaphor might assist subjects to form bizarre mnemonic images). This third experiment therefore added a third condition to those in experiment B, in which no metaphor at all was given (as in the first experiment).

For this experiment, a further two diagrams were created, illustrating the computational concepts of remote execution and selection – once again using everyday examples rather than programming terminology.

## 4.1. Method

Twelve subjects were recruited from the APU volunteer panel. All were programming novices. As in experiment B, subjects were given a balanced number of diagrams for each metaphor condition: systematic, nonsense, and no metaphor at all. They answered comprehension questions for examples of every diagram, and completed incomplete diagrams by writing in missing labels. The test in which incomplete diagrams were completed by drawing was omitted from this experiment, as it had been particularly unaffected by metaphor use, and may therefore have masked the benefits of metaphor in the other tasks.

Material was again presented to subjects in booklet form. Subjects wrote directly in the booklet, and used a stopwatch to record the amount of time they spent working on each page. As in experiment B, performance was measured using speed with which participants completed the tasks, and accuracy in the tasks. Accuracy scores were again normalised for each task, so that treatment effects could be compared using scores on different tasks.

## 4.2. Results

The hypothesis for this experiment was that performance with no metaphor would be intermediate between that in the systematic metaphor and nonsense metaphor cases. In fact, there was no significant difference in speed between the three cases,  $F(2,22)=0.31$ ,  $p=.735$ . Furthermore, the mean accuracy in the no-metaphor case was actually slightly higher overall than in either of the other cases, as shown in table 1. This difference in accuracy was not significant either, however,  $F(2,126)=0.37$ ,  $p=.694$ .

**Table 1. Mean accuracy (%)**

Metaphor type	Combined mean score
Nonsense	46.2
None	51.8
Systematic	47.4

Why might the absence of a metaphor improve performance? Participants might have noticed that the non-metaphorical explanations were shorter than the others in the booklet, and realised that some systematic motivation for the choice of symbols had been hidden from them. They may well have constructed their own explanatory metaphor at this point. A self-generated explanation may even be a superior mnemonic to the metaphor provided in other explanations. In order to test this, we compared the time that participants spent reading the diagram explanation. Reading time would normally be shorter in the condition with no metaphor, as the text is shorter. If participants were constructing their own explanations, reading time should be longer for the condition with no metaphor.

Reading times for each metaphor condition are shown in table 2. Reading time is in fact shorter for the no-metaphor condition. It is longest for the nonsense condition (as in the case of experts in experiment B), suggesting that participants spend more time trying to make sense of the nonsense metaphor.

**Table 2. Reading time**

Metaphor Type	Reading Time (s)	SD
Nonsense	126.2	13.9
None	95.6	5.6
Systematic	104.9	12.1

## 4.3. Discussion

This experiment replicates the results of experiment B, in which no clear benefit was found as a result of providing an explicit instructional metaphor. Not only was no difference found between systematic metaphor and a nonsense metaphor, but providing no metaphor at all seems to be just as beneficial for understanding visual representations of simple programming constructs.

## 5. Conclusions

Visual programming languages have often been described as providing usability advantages for inexperienced programmers because they are easier to learn than textual languages [3][41]. The experiments in this study have investigated one hypothesis regarding the usability of visual languages – that they can assist the user to understand a virtual machine through some metaphorical presentation of the machine in diagrammatic form.

The experimental results have not supported that hypothesis, however. The experiments described in this paper tested the hypothesis by making instructional metaphors more or less available to inexperienced programmers, and evaluating the resulting changes in usability by comparison to the performance of experienced programmers. In experiment A, the metaphor was conveyed by making the elements of the diagram more pictorial, while in experiments B and C the metaphor was explicitly described in instructional material. In neither case did the provision of the metaphor result in appreciable performance improvements relative to more experienced programmers, despite the fact that the computational concepts included in the diagrams of experiment 2 were heavily disguised, and the fact that the diagrams were equally novel to all subjects.

This negative result is not what we expected when we commenced the study. It cannot be criticised on the basis that the measures of usability were ineffective, that the diagrams and metaphors were simply poorly designed, or that the metaphors were ignored by experimental subjects, because the measures of performance were successful in



distinguishing between expert and novice performance, yet they failed to demonstrate the substantial usability benefits that are often claimed for metaphor. We confidently expected to see those benefits when this study was initiated. This expectation did lead to experimental designs with smaller numbers of subjects than would be needed to discount any effect at all – but the fact that the effects, if any, were so small over multiple experiments casts doubt on the central importance of metaphor.

These results do, moreover, follow a pattern consistent with other empirical research projects in HCI, which have failed to find significant usability benefits when metaphors are compared to non-metaphorical interfaces [32]. These studies generally view their findings in an optimistic light (for example, as resulting from individual differences in the experimental sample, as in a study by Rohr [33]), and do not question the central assumption regarding the usability value of metaphor (e.g. in a study by Simpson and Pellegrino [34]).

The situation is also confused by the fact that studies demonstrating the advantages of direct manipulation interfaces (e.g. those by Benbasat and Todd [1] or Davis & Bostrom [10]) are often quoted in support of the value of metaphor. Such studies do not in fact separate the manipulation of metaphor from other interface characteristics. We do not deny that graphical user interfaces in general are valuable, and that direct manipulation in particular provides clear benefits – we simply suggest that these benefits might have been wrongly attributed to instructional metaphor. If those interfaces were tested with and without explanatory metaphors, it seems possible that direct manipulation would be equally effective with no associated metaphor.

In the case of visual programming languages, we think that these results do at least suggest caution regarding the application of metaphor to visual languages. We recognise that there are successful visual languages that include extended metaphors (Kahn's ToonTalk [18] is a prominent example), so the use of an explicit metaphor in an instructional context can obviously be valuable. However our results show that usability benefits derived from improved learnability are not necessarily automatic. Our recommendations for future research are firstly, that metaphor should not be treated as an necessary criterion when designing visual representations, and secondly that further empirical investigations should be made of those VPLs which successfully incorporate explicit metaphors, in order to determine whether the usability of the language really derives from the metaphor, or from some other source. Our continuing investigations suggest that some alternative contributing factors include level of attention given to the metaphor during training, additional cognitive load involved in interpreting a systematic metaphor, and the well-known mnemonic benefits of combining pictorial images with verbal cues (even in the absence of any systematic pictorial metaphor).

## Acknowledgements

Alan Blackwell's contribution to this research was funded by a collaborative studentship from the Medical Research Council and Hitachi Europe Ltd. He is grateful to the Advanced Software Centre of Hitachi Europe for their support.

## References

- [1] I. Benbasat and P. Todd, "An experimental investigation of interface design alternatives: icon vs. text and direct manipulation vs. menus". *International Journal of Man-Machine Studies*, 38:369-402, 1993.
- [2] A. F. Blackwell, "Metaphor or analogy: How should we see programming abstractions?" In P. Vanneste, K. Bertels, B. De Decker & J.-M. Jaques (Eds.), *Proceedings of the 8<sup>th</sup> Annual Workshop of the Psychology of Programming Interest Group*, pp. 105-113, 1996.
- [3] A. F. Blackwell, "Metacognitive theories of visual programming: What do we think we are doing?" *Proceedings IEEE Symposium on Visual Languages*. Los Alamitos, CA: IEEE Computer Society Press, pp. 240-246, 1996.
- [4] A. F. Blackwell, K. N. Whitley, J. Good, and M. Petre, "Cognitive Factors in Programming with Diagrams". To appear in *Artificial Intelligence Review*, special issue on Thinking with Diagrams.
- [5] D. A. Boehm-Davis, , J. E. Fox and B. H. Philips, "Techniques for exploring program comprehension". In W. D. Gray & D. A. Boehm-Davis (Eds.), *Empirical Studies of Programmers: Sixth Workshop*. Norwood, NJ: Ablex, pp. 3-38, 1996.
- [6] S. Booth, "The experience of learning to program. Example: recursion". *Proceedings of the 5<sup>th</sup> Workshop of the Psychology of Programming Interest Group*: INRIA, pp. 122-145, 1992.
- [7] J. J. Cañas, , M. T. Bajo and P. Gonsalvo, "Mental models and computer programming". *International Journal of Human-Computer Studies*, 40(5):795-811, 1994.
- [8] J. M. Carroll and J. R. Olson, "Mental models in human-computer interaction". In M. Helander (Ed.), *Handbook of Human-Computer Interaction*. Elsevier., pp. 45-66, 1988.
- [9] Y. S. Chee, "Applying Gentner's theory of analogy to the teaching of computer programming". *International Journal of Man Machine Studies*, 38(3):347-368, 1993.
- [10] S. A. Davis and R. P. Bostrom, "Training end users: An experimental investigation of the roles of the computer interface and training methods". *MIS Quarterly*, 17:61-85, 1993.
- [11] B. du Boulay, T. O'Shea and J. Monk, "The black box inside the glass box: Presenting computing concepts to novices". *International Journal of Man-Machine Studies*, 14(3):237-249, 1981.
- [12] R. E. Eberts and K. P. Bittianda, "Preferred mental models for direct manipulation and command-based interfaces". *International Journal of Man Machine Studies*, 38(5):769-786, 1993.

- [13] M. Eisenberg, M. Resnick and F. Turbak, "Understanding procedures as objects". In G.M. Olson, S. Sheppard & E. Soloway (Eds.), *Empirical Studies of Programmers: Second Workshop*. Norwood, NJ: Ablex, pp. 14-32, 1987.
- [14] M. Eisenstadt, J. Breuker and R. Evertsz, "A cognitive account of 'natural' looping constructs". In B. Shackel (Ed.), *Human-Computer Interaction – Interact '84*. Amsterdam: Elsevier, pp. 455-460, 1984.
- [15] O. Esteban, S. Chatty and P. Palanque, "Whizz'ed: A visual environment for building highly interactive software". In K. Nordby, P. Helmersen, D.J. Gilmore & S.A. Arnesen (Eds.), *Human Computer Interaction: Interact '95*. London: Chapman & Hall, pp. 121-126, 1995.
- [16] E. P. Glinert, "Nontextual programming environments". In S-K. Chang, (Ed.), *Principles of Visual Programming Systems*. Prentice-Hall, pp. 144-232, 1990.
- [17] A. Jones, "How novices learn to program". In B. Shackel (Ed.), *Human Computer Interaction – INTERACT'84*. North Holland: Elsevier, pp. 777-783, 1984.
- [18] K. Kahn, "Seeing systolic computations in a video game world". *Proceedings IEEE Symposium on Visual Languages*. Los Alamitos, CA: IEEE Computer Society Press, pp. 95-101, 1996.
- [19] R. E. Mayer, "Different problem-solving competencies established in learning computer programming with and without meaningful models". *Journal of Educational Psychology*, 67(6):725-734, 1975.
- [20] R. E. Mayer, "From novice to expert". In M. Helander (Ed.), *Handbook of Human-Computer Interaction*. Elsevier, pp. 569-580, 1988.
- [21] R. E. Mayer and V. K. Sims, "For whom is a picture worth a thousand words? Extensions of a dual-coding theory of multimedia learning". *Journal of Educational Psychology*, 86(3):389-401, 1994.
- [22] Microsoft Corporation, *The Windows interface guidelines for software design*. Redmond, WA: Author, 1995.
- [23] B. A. Nardi and C. L. Zamer, "Beyond models and metaphors: visual formalisms in user interface design". *Journal of Visual Languages and Computing*, 4(1):5-33, 1993.
- [24] R. Noble, "Preferential use of examples by novices learning Prolog". In *Proceedings 5<sup>th</sup> Workshop of the Psychology of Programming Interest Group*. INRIA, pp. 146-158, 1992.
- [25] D. A. Norman, *The psychology of everyday things*. New York, Basic Books, 1988.
- [26] D. A. Norman, "Cognitive artifacts". In J.M. Carroll (Ed.), *Designing Interaction: Psychology at the Human-Computer Interface*. Cambridge: Cambridge University Press, pp. 17-38, 1991.
- [27] J. Pane, "A programming system for children that is designed for usability". In C. Kann (Ed.), *Proceedings of the First ESP Student Workshop*, pp. 15-22, 1997.
- [28] J. F. Pane and B. A. Myers, *Usability issues in the design of novice programming systems*. School of Computer Science, Carnegie Mellon University. Technical Report CMU-CS-96-132, 1996.
- [29] S. J. Payne, "Metaphorical instruction and the early learning of an abbreviated-command computer system". *Acta Psychologica*, 69:207-230, 1988.
- [30] S. J. Payne, "A descriptive study of mental models". *Behaviour and Information Technology*, 10(1):3-21, 1991.
- [31] S. J. Payne, H. R. Squibb and A. Howes, "The nature of device models: the yoked state space hypothesis and some experiments with text editors". *Human-Computer Interaction*, 5(4):415-444, 1990.
- [32] K. Potosnak, "Do icons make user interfaces easier to use?" *IEEE Software*, May 1988, pp. 97-99. .
- [33] G. Rohr, "How people comprehend unknown system structures: Conceptual primitives in systems' surface representations". In P. Gorny & M.J. Tauber (Eds.), *Visualization in Programming*. Lecture Notes in Computer Science Vol. 282. Berlin: Springer-Verlag, pp. 89-105, 1987.
- [34] H. K. Simpson and J. W. Pellegrino, "Descriptive models in learning command languages". *Journal of Educational Psychology*, 85(3):539-550, 1993.
- [35] A. Sutcliffe and U. Patel, "3D or not 3D: Is it nobler in the mind?" In M.A. Sasse, R.J. Cunningham & R.L. Winder (Eds.), *People and computers XI: Proceedings of HCI'96*. London: Springer-Verlag, pp. 79-94, 1996.
- [36] M. J. Tauber, "On visual interfaces and their conceptual analysis". In P. Gorny & M.J. Tauber (Eds.), *Visualization in Programming*. Lecture Notes in Computer Science Vol. 282. Berlin: Springer-Verlag. pp. 106-123, 1987.
- [37] J. Taylor, "Analysing novices analysing Prolog: What stories do novices tell themselves about Prolog?" *Instructional Science*, 19:283-309, 1990.
- [38] J. D. Tenenber, "Virtual machines and program comprehension". In P. Vanneste, K. Bertels, B. De Decker & J.-M. Jaques (Eds.), *Proceedings of the 8<sup>th</sup> Annual Workshop of the Psychology of Programming Interest Group*, pp. 60-82, 1996.
- [39] M. Treglown, "Qualitative models of user interfaces". In G. Cockton, S.W. Draper & G.R.S. Weir (Eds.), *People and computers IX: Proceedings of HCI'94*. London: Springer-Verlag, pp. 261-272, 1994.
- [40] S. Watt, "Syntonicity and the psychology of programming". In J. Domingue & P. Mulholland (Eds.), *Proceedings of the Tenth Annual Meeting of the Psychology of Programming Interest Group*, pp. 75-86, 1998.
- [41] K. N. Whitley and A. F. Blackwell, "Visual programming: the outlook from academia and industry". In S. Wiedenbeck & J. Scholtz (Eds.), *Proceedings of the 7th Workshop on Empirical Studies of Programmers*, pp. 180-208, 1997.
- [42] D. M. Willows, "A picture is not always worth a thousand words: pictures as distractors in reading". *Journal of Educational Psychology*, 70:255-262, 1978.
- [43] P. Wright, R. Milroy and A. Lickorish, "Static and animated graphics in learning from interactive texts". *European Journal of Psychology of Education*, 14(1) Special issue on Visual Learning with New Technologies 1999.
- [44] R. M. Young "The machine inside the machine: Users' models of pocket calculators". *International Journal of Man-Machine Studies*, 15(1):51-85, 1981.