THE 'RIGHT' TOOL FOR THE TASK: AN INVESTIGATION OF EXTERNAL REPRESENTATIONS, PROGRAM ABSTRACTIONS AND TASK REQUIREMENTS

Judith Good
Department of Artificial Intelligence
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN, Scotland
judithg@aisb.ed.ac.uk

KEYWORDS: program abstractions, graphical representations, task requirements.

ABSTRACT

The pilot study described in this paper considers the utility of diverse types of graphical representations of programs for use by novices when reasoning about the program's behaviour and/or states. The study tests the hypothesis of *information congruency*: *i.e.* that performance on a task improves when the type of information highlighted in a given representation is congruent with the information required by the task. It also looks at the effect of differing presentation type on performance, and the interaction between information congruency and presentation type. Finally, it presents a qualitative analysis of the types of strategies used and the misunderstandings which occur when attempting to navigate through the diagram. The results of this preliminary study tend not to support the information congruency hypothesis and seem to suggest that other factors are at work. Likewise, although there is a trend for presentation to affect performance, the results are not conclusive. The general outcome of the study suggests the need for a more formal analysis of the *processes* involved in reasoning over data flow and control flow representations.

1. INTRODUCTION

The importance of representation in problem solving is well established. As Pennington and Grabowski (1990) note, the psychological literature documents the fact that the particular representation[1] of a problem which is constructed has a direct influence on the difficulty of problem solving and on the range of solutions that will be considered.

Although Simon (1969) states that all of problem solving can be characterised as changes of representation and that problem solving itself consists of representing a problem so that its solution becomes apparent, other factors are likely to be involved. Cox and Brna contend that the "ideal" representation at any given time will vary according to an individual's personal experience, cognitive style preferences and the requirements of the task (Cox & Brna, 1995). Sometimes even a single task, for example, solving an analytical reasoning problem, will require more than one representation if it is to be efficient. In areas such as mathematics and science, Schwarz and Dreyfus (1993) maintain that a single representation cannot generally represent all of the formal properties of a particular concept, and that multiple representations are needed, each of which highlights a specific aspect.

While it would seem that experts are able to use multiple representations and move between them with relative ease so as to exploit the inherent advantages of each one, novices have trouble in doing so (Tabachneck, Leonardo, & Simon, 1994). Tabachneck *et al* maintain that not only do novices not seem to do so spontaneously, but show ineffective use and integration of the representations when required to do so. In the area of programming, Anderson (1988) cites the difficulty that many novice

[1] The authors explicitly define "representation" as the mental conception of the problem and/or an external representation, which may be graphical or sentential.

programmers have with recursion as stemming from an inability to change their "view" of recursion: for example, novices will focus on recursion as a type of control flow rather than on the transformations it produces on data objects.

However, how does one select the "best" representation to use at any given time? Cox and Brna (1995) report that the issue of representation selection has been the subject of much "folk wisdom" but little empirical work. Systems which incorporate multiple representations don't always address the issue either: there often seems to be an implicit assumption that providing representations which the user is thought to need will allow the user to choose the type of information required when required (Brayshaw, ress), without considering how the user determines his or her requirements in the first place.

As representations highlight some information at the expense of other information (see for example (Pennington, 1987; Schwarz & Dreyfus, 1993)), it would seem reasonable to assume that the representation which highlights the information required by the task in question (in the sense that it requires less inference in order to extract the information needed) should allow for more effective problem solving than one which doesn't. For example, in programming, a data flow representation should provide more assistance in answering a question requiring data flow information, even if this information can in principle be derived from a control flow representation.

On the other hand, the same information may be represented in very different ways and it is not always clear why one representation is easier to use or more effective than another one. Although it could be said that this is due to the fact that the representations are *informationally* but not *computationally* equivalent (Larkin & Simon, 1987), this distinction is not always useful in the sense that the computational properties of the diagram may not necessarily lie with the diagram itself, but in the interaction between the user and the diagram.

Thus, if this is so, it is not at all clear how to decide when two representations are actually computationally equivalent for a given user in a specific context. If the notion of computational equivalence is to retain any value, it is necessary to factor in the computational effects of the interaction between the user and the "equivalent" representations.

There has been some significant recent work carried out to examine graphical versus sentential modalities by comparing and contrasting visual and textual languages (*e.g.* (Cunniff & Taylor, 1987; Green, Petre, & Bellamy, 1991; Green & Petre, 1992; Pandey & Burnett, 1993)), however, the emerging picture is quite complicated. Visual languages often have textual content and textual languages have visual properties; users have to learn to read visual languages; different levels of experience affect how visual (and textual) languages are used; and the nature of the task is a major factor. The hypothesis that visual programming languages are better than textual ones is not easily supported. A slightly more sophisticated hypothesis is that certain tasks lend themselves to specific ways of abstracting programs and to specific ways of representing that information.

In this paper we investigate two graphical ways of abstracting information in programs and two specific ways of presenting this information. We address two hypotheses which relate these issues to task performance. Briefly stated, we hypothesise that enhanced performance will result from situations in which the information extracted from a program corresponds to that required by a task. Furthermore, we hypothesise that the way in which this information is presented will also have an effect on task performance. More detailed statements of these hypotheses will be made below. The main conclusion which can be drawn from this study is that the relationship between information, representation, task and user is complex, and requires further work in order to tease out the subtle interactions between factors.

2. THE STUDY

The pilot study described here investigates the notion of *congruence* as a task facilitator using diagrammatic representations of programs based on two of the program abstractions described in (Pennington, 1987): data flow and control flow diagrams. According to Pennington, these program abstractions, which also include ones relating to program functionality and conditionalised action, highlight some information present in the program text at the expense of other information. However, this information is nonetheless derivable, *e.g.* by looking for repeated data object names, data flow can be derived from a control flow abstraction.

We define congruence as a situation in which the information being highlighted in the representation is the same as that required by the task. For example, a control flow diagram and a question about events which precede or follow from a given state are said to be congruent. The same question accompanying a data flow diagram is said to be incongruent. The notion of congruency may be applicable to all four of the abstractions discussed in (Pennington, 1987), although only two were tested here.

The idea of congruency versus incongruency shares similarities with Gilmore and Green's match-mismatch conjecture (Gilmore & Green, 1984), which, at a high level of abstraction, makes the claim that "the structure of a notation affects the ease with which information can be extracted both from the printed page and from recall" (p. 47). Gilmore and Green were investigating the match between the notation and the information required by the task, where the notation was either procedural or declarative, and the information required by the task was either sequential or circumstantial. In the case of the current study, both the notation and the information required by the task are categorised in terms of data flow and control flow.

The idea is also similar to that of cognitive fit, which was investigated in various domains by Vessey and her colleagues (Vessey, 1991; Sinha & Vessey, 1992). Cognitive fit refers to the match between the information emphasised in the problem representation and the task. It facilitates problem solving by leading to the use of similarly matching problem solving processes (and a matching mental representation). The idea of cognitive fit was applied to domains such as the graphs and tables literature (Vessey, 1991) and an empirical investigation into recursion and iteration[2].

A second topic of interest of the current study concerns variations in the way information is represented, in this case in the form of graphs or trees[3], and how this too may influence problem solving, particularly for recursive problems such as those used in the study. Various hypotheses were entertained as to their effect: trees may represent a larger search space in the sense that each recursive call, rather than being represented implicitly as a "loop" in the graph, generates an additional level in the tree. Thus, even if the information contained in the tree is replicated to a certain extent, recovering from erroneous paths may be more time-consuming than for graphs. On the other hand, graphs (and particularly cyclic graphs such as the ones used in the study), while being more compact visually, lack the advantage of trees in that they do not provide a visual record of states previously visited in the form of a path from the root to the current node.

Finally, the study was designed so as to gather qualitative data on how people navigate through diagrams. Although claims have been made for visual programming languages (Shu, 1988; Golin, 1991), they have not always been supported by empirical evidence. Green *et al* (1991, 1992) highlight some of the difficulties involved in using graphical representations of programs; this study aims to provide further information.

2.1. Hypotheses

Two main hypotheses were investigated in this study: the information congruency hypothesis and the presentation hypothesis.

The information congruency hypothesis was formulated as follows: when the question type (data flow or control flow) is congruent with the information content of the diagram (again, data flow or

[2] Both the match-mismatch conjecture and the cognitive fit hypothesis will be considered further in Section 2.7.

[3] Note that in terms of data flow, trees are something of a misnomer in the sense that any operation requiring more than one data object at the outset will have more than one root. A more correct name for the data flow trees would be *acyclic* graphs (as distinguished from *cyclic* graphs, such as the data and control flow graphs used in the experiment). However, the similarity between the two terms proved to be rather confusing, therefore the terms *tree* and *graph* were maintained.

control flow) then performance will be enhanced, as measured by a lower response time and a fewer errors. Conversely, an incongruency between the question type and the information content of the diagram (*e.g.* a data flow question coupled with a control flow diagram) should result in a decrease in performance, as manifested by a greater number of errors and an increase in time taken to complete the task.

The second hypothesis concerned the presentation type, *i.e.* the particular way in which the information content is externally represented. In this study, two types of presentation were investigated, trees and graphs. It was expected that diagram type would affect performance, with one or the other favouring problem solving, measured again in terms of speed and accuracy. However, this was a bidirectional hypothesis: as described above, there were opposing tensions as to the advantages of each type of diagram and it was unclear how these would interact. In addition, we are unaware of similar studies which compare these two types of representation in the area of programming and thus did not have empirical or experimental results on which to base our predictions.

The final objective of the study concerned the possible interaction between congruency and presentation type: we were interested in determining the effect, if any, of combining a particular type of representation (data or control) with its physical manifestation (tree or graph).

2.2. Subjects

Ten subjects took part in the experiment. All subjects except one were MSc students in either Artificial Intelligence or Cognitive Science at the University of Edinburgh[4]. Subjects were recruited from amongst those taking the Programming in Prolog course which was running that term because the study focussed on recursive problems, hence it was important that subjects have a certain level of experience in a language which relies heavily on recursive constructs. All ten subjects had taken, or were in the eighth week of, a ten-week introductory Prolog course which included extensive use of recursive programs.

2.3. Design

The experiment was a two-factor randomised mixed design, with each factor having two levels. The variables being manipulated were the information content of the diagram, in the form of control flow or data flow diagrams, which was a between subjects factor, and presentation type, in the form of trees or graphs, which was a within subjects variable. Subjects were randomly assigned to either the control flow or the data flow condition, and the presentation order of trees and graphs was counterbalanced across and within subjects so as to control for order effects.

Although a pure within subjects design has the advantage of minimising the effect of any individual peculiarities, doing so would have made the experiment overly long, a factor which would probably have outweighed any potential benefits. Additionally, a between subjects design was unfeasible given the small number of subjects in the pilot study, but is planned for further studies (see Section 2.7. for details).

2.4. Materials

The materials used consisted of a paper based pre-test, a Hypercard stack which incorporated the practice materials and the experimental stimulus, and a screen recording utility to record the subjects' interactions with the system[5].

The Pre-Test. The paper and pencil pre-test consisted of five multiple choice questions designed to test subjects' understanding of both tail and non-tail recursion. The questions used a textual representation of the code, and required the subjects to recognise correct versions of a particular program,

[4] The remaining subject had previously studied for an MSc in Artificial Intelligence.
[5] Farallon Inc.'s 'ScreenRecorder' utility, part of the 'MediaTracks' package.

determine the behaviour of a standard but unnamed piece of code, correct a piece of buggy code, determine and distinguish between the effects of two different types of recursive list building, and predict the invocation order of lines of code, given a particular query.

In addition to the recursion questions, the pre-test also included a question on the subjects' knowledge of various programming languages, *i.e.* which languages they had learned, how long they had used them, and to what extent (*e.g.* "took a course on it", "used extensively in my job", "used off and on").

The Experiment. The experiment was implemented as a Macintosh Hypercard stack, and consisted of a familiarisation stage and the experimental trials. The first screen gave a short description of the various stages of the experiment. This was followed by the familiarisation session, which began with a screen showing a graph diagram, and a description of each of the component parts of the diagram in terms of their shape, meaning and function. The subsequent screen showed the same diagram and a sample question similar to those asked in the experiment proper. This was repeated for tree diagrams, *i.e.* each subject saw a tree diagram accompanied by an explanation of its component parts, followed by a sample question on the same diagram.

Following this were the experimental trials: after a short textual description of a piece of code in terms of input, output and a concrete example of both, subjects were presented with either a tree or a graph representation of this code along with a multiple choice question. The question was either congruent or incongruent with the diagram. The next stimulus showed the same diagram and a new question: if the first question had been congruent, this second question would be incongruent, and vice versa. This sequence of textual description, followed by diagram plus question, then diagram plus new question was repeated for a second program. Thus each subject answered a total of four questions, two per diagram.

Trials differed for control flow and data flow groups in that the control flow group used a control flow tree and a graph, while data flow subjects used a data flow tree and a graph. For both groups, the order of presentation of tree and graph was counterbalanced, as was the order in which the problems were presented (note that both groups received the same questions: a data flow question and a control flow question for each diagram).

The diagrams themselves depicted short recursive problems. We decided to use recursive problems as the focus of this study because, firstly, the psychology of programming literature documents the difficulty involved in both learning and using recursion (*e.g.* (Anderson et al., 1988; Er, 1984; Kahney, 1989)). Secondly, the nature of recursion is such that difficulty is not a function of size: small, compact programs are often more difficult to understand, and we felt that a graphical representation of such programs would provide a good opportunity for observing the types of difficulty discussed in (Green et al., 1991; Petre & Green, 1993).

The programs used were `max` and `position` (Appendix A contains the code for these programs). The former takes a list of numbers and recursively scans it to find the maximum number in the list while the latter takes an element, and a list containing that element, and returns the position of that element in the list. The problems were chosen because, although they are not uncommon, they both have particular properties: `max` involves the swapping of values between variables, while a non-tail recursive version of `position` was chosen as results from (Kurland & Pea, 1983) show that performance on tail recursive problems do not allow us to distinguish between a correct model of recursion and a model of recursion as iteration.

Figure 1 shows an example of a control flow graph for the `position` program. The control flow graphs were based to some extent on a flow chart formalism, with nodes representing tests and actions, and arcs representing the flow of control between boxes. Note however that the meaning of the particular shapes differed from conventional flowcharts, *e.g.* an ellipse was used to denote tests, while a rounded box indicated variable assignment.

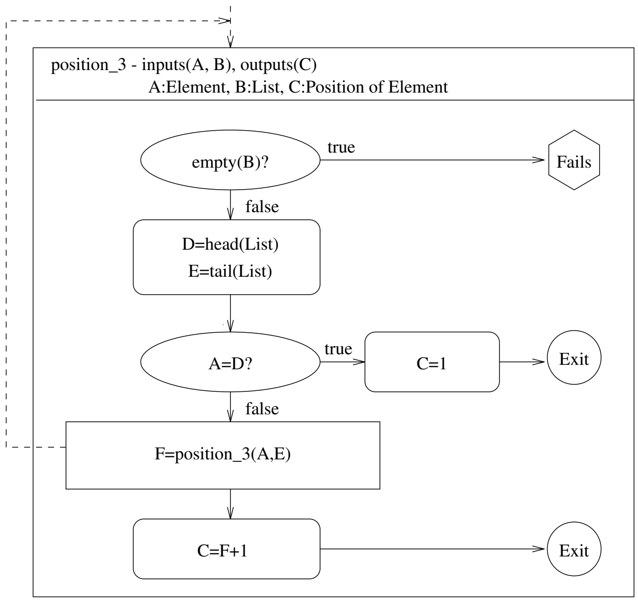

Figure 1: Control Graph Representation for position

Figure 2 shows a control flow tree for the same program. The control flow tree is based on AND/OR tree representations, with each level in the tree denoting an individual recursive call. At any given node, the path taken from the root node to that point can be traced. Note that variable renaming is not made explicit for each recursive call.

Figure 3 shows a data flow graph for the same program. The data flow formalism followed the basic description of visual data flow languages in (Hils, 1992) and (Davis & Keller, 1982), in which nodes represent functions and arcs represent the flow of data between those functions. The 'pure' data flow model was augmented with selectors, which allow a true or false data token to be propagated along the output arc, and distributors, which use a true or false token in order to decide which output arc to choose for the data.

Finally, Figure 4 gives an example of a screen presented to the user, showing a data flow tree for `position` with a control flow (incongruent) question.

2.5. Method

Subjects were first given the pre-test and allowed to spend as much time on it as they wished.

Subjects then worked through the experiment, again at their own pace. All subject/system interactions were logged and time-stamped using the ScreenRecorder utility.

When subjects finished the experiment, they were asked to view the screen recording of their session and talk the experimenter through it, *i.e.* on the basis of the mouse movements, explain what they were doing at any given time. Subjects used the ScreenRecorder playback facilities to fast forward and/or pause the recording as needed, and this part of the experiment was audio taped.

2.6. Results

The design of the study permitted the examination of two main effects, information congruency and presentation type, and the interaction between them to be examined. Following a brief discussion of the pre-test results, this section looks at results in terms of the main effects and their interaction, and then goes on to discuss some qualitative results from the retrospective protocol analysis.

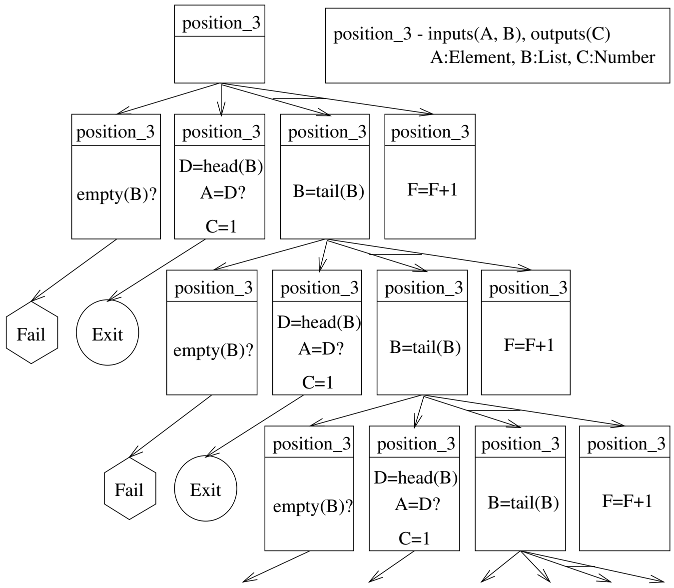

Figure 2: Control Tree Representation for `position`

Pre-test. The median score on the pre-test was 4 points (out of a possible 5 points – one point for each question) with 80% of scores falling between 3 and 5 points. With the exception of one question, which all subjects answered correctly, incorrect responses were evenly distributed over questions and multiple choice options.

Practice Session. All subjects saw two questions during the practice session (one accompanied by a tree diagram, and one by a graph diagram). Although these questions were not part of the experiment per se, the results were compared so as to ensure that that were no major differences between the data and control flow groups, and are summarised below:

Groups	Mean Speed (in seconds)	Accuracy
Data Flow	259.8	7 correct, 3 no answer
Control Flow	274.2	8 correct, 2 incorrect

Table 1: Results of Practice Questions

Information Congruency. The congruency hypothesis, *i.e.* the hypothesis that performance would be improved when the information required by the task matched that present in the diagram, was examined in terms of both response latency and error rates[6].

Each question was scored simply as either correct (1 point) or incorrect (0 points), and it was expected that the error rate (in terms of the percentage of incorrect answers) would be lower in congruent situations. Overall, the error rate for congruent questions was 50%, as opposed to 35% for

[6] Note that these were correlated to check for speed-accuracy trade-offs, however the correlation was not significant (Spearman rank-order correlation coefficient, r_s=.051, ns).

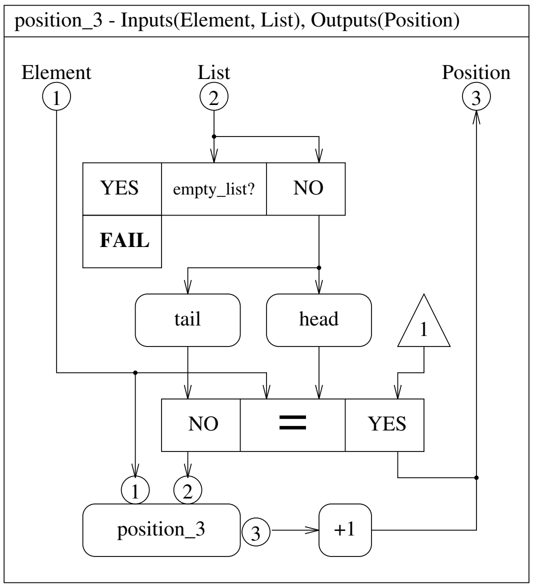

Figure 3: Data Graph Representation for `position`

incongruent situations. This difference between conditions goes in the opposite direction to that predicted (*i.e.* the error rate was lower when the information type and question were *not* congruent). As the accuracy data was deemed to be ordinal rather than interval, non-parametric statistics were employed. Using a Wilcoxon signed ranks test, the difference between conditions was not significant (T^+=24, N=8, ns).

Mean response latency was 182 seconds for congruent questions and 165.5 seconds for incongruent questions, a trend which again goes in the opposite direction to that predicted. Time data was screened to ensure that requirements for parametric statistics were met. The difference between conditions was not significant (t-test, t=.587, ns).

Figure 5 shows the plotted means for speed and error rates (with 95% confidence intervals).

Figure 6 gives the mean response time for individual subjects on both congruent and incongruent questions; seven out of ten subjects responded more quickly in the incongruent condition.

Thus, the main hypothesis, that congruency between question type and information content in the diagram would lead to better performance, was not supported. In fact, there is a tendency in the opposite direction, in the sense that congruent situations were associated with a longer solution time and a greater number of errors.

Presentation Type. The second hypothesis concerned the effect of varying presentation type on performance. As each subject received two questions accompanied by graphs and two accompanied by trees, we can compare their scores on each. The error rate for tree questions was 55% as opposed to 30% for graph questions. This difference was not, however, significant (Wilcoxon signed ranks test, T^+=24, N=7, ns).

In terms of speed, the mean time taken to answer questions using trees was 198.9 seconds, as opposed to 148.7 for questions using graphs. Again, this result is not statistically significant (t-test, t=1.277, ns) due to large variation between subjects and within groups.

Figure 7 shows the plotted means for speed and error rates (with 95% confidence intervals).

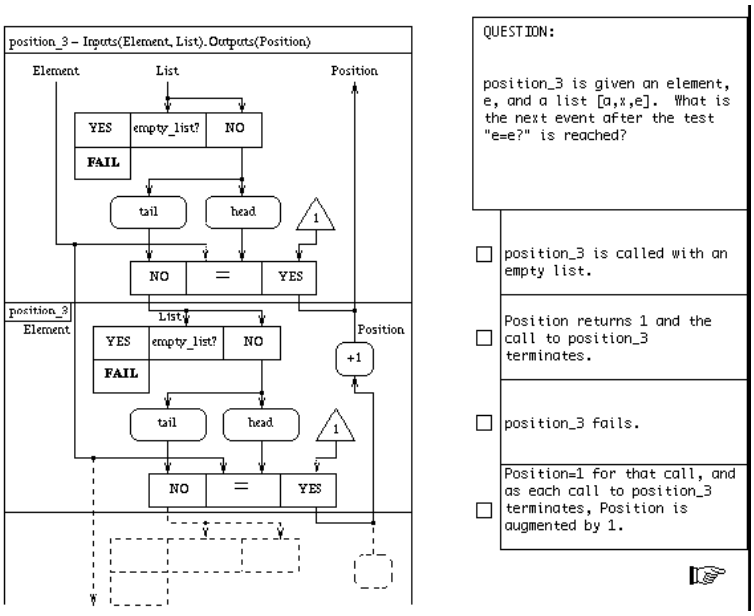

Figure 4: Data Flow Tree for `position` with Control Flow Question

Figure 8 shows the mean response time for individual subjects using trees and graphs; for six out of ten subjects, the use of graphs resulted in a lower response time.

The effect of presentation type on accuracy is shown in Table 2. Each subject received two questions accompanied by a graph and two accompanied by a tree. Thus, for each presentation type, scores range from 0 (neither question answered correctly) to 2 (both questions answered correctly). Table 2 shows a strong tendency for high scores (*i.e.* both questions correct) to occur when the questions are accompanied by graphs (5 out of 10 subjects answered both questions correctly using graphs as opposed to only 1 out of 10 using trees). Possible reasons for this trend will be discussed in Section 2.7.

In summary, the results suggest that graphs are associated with more accurate responses and an improved response time, although the results are not significant.

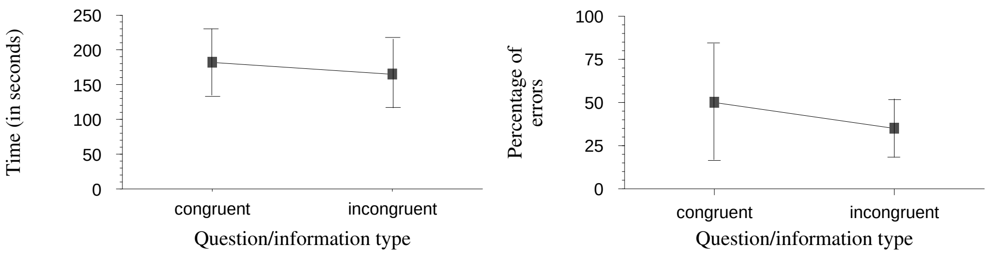

Figure 5: Response latency and percentage of errors for congruent and incongruent questions

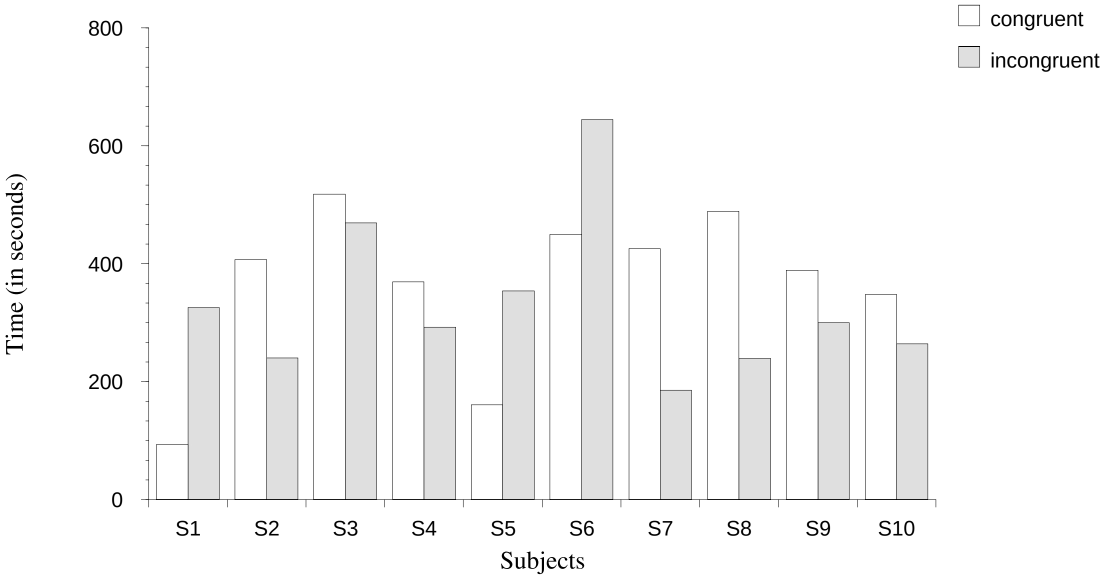

Figure 6: Response Time per Subject: Congruent versus Incongruent

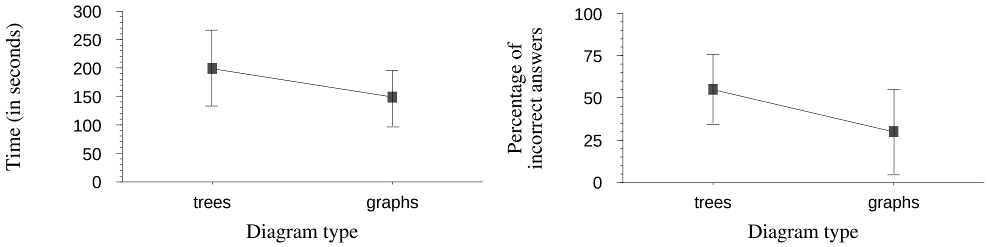

Figure 7: Response latency and percentage of errors for use of trees versus graphs

Interaction between main effects An analysis of variance on the time data was run to determine the significance, if any, of the main effects (group, presentation type and congruency) and their interactions. None of the main effects or interactions were significant due to low sample sizes and high error variance. Nevertheless, rank ordering the size of the effects suggests that the effects of presentation type were more consistent than the effects of information congruency.

Qualitative results. This section presents the qualitative data obtained from the retrospective verbal protocols and looks at both the strategies which subjects used and also at misunderstandings which arose[7].

Verbal protocols were taken from subjects as they explained their screen recording to the experimenter after the experiment. The analysis of the protocol looked at two types of information: the *strategies* used by subjects as they reasoned using the diagram, and the *misunderstandings* which

[7] Like Mulholland (1994), the term *misunderstanding* will be used rather than *misconception* as it is believed that problems observed were due in part to an unfamiliarity with the notation as opposed to any deep-seated conceptual difficulties.

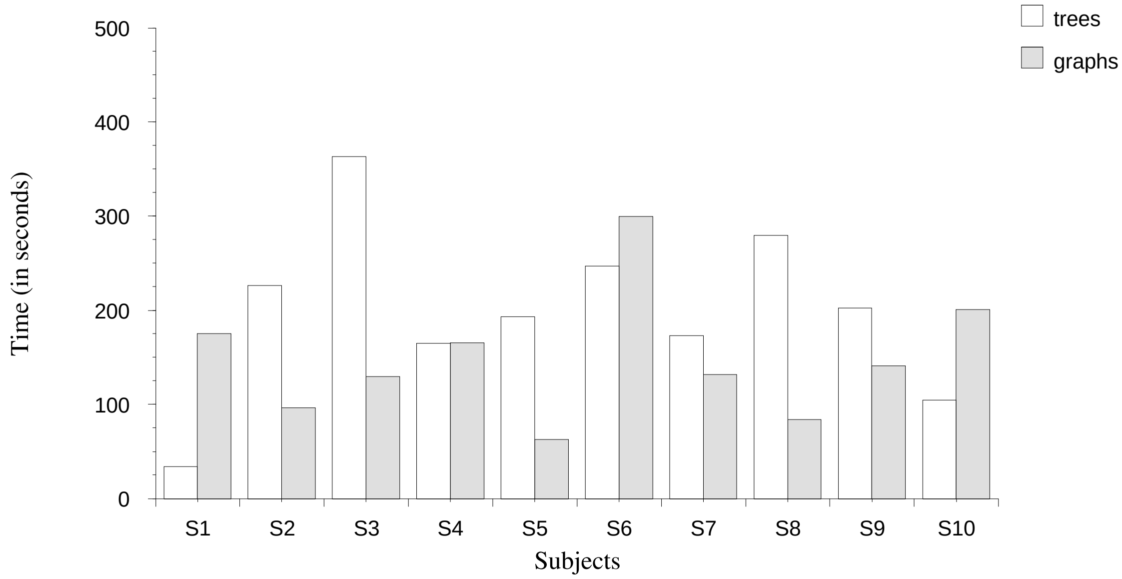

Figure 8: Response Time per Subject: Trees versus Graphs

Presentation Type	**Score**		
	2	**1**	**0**
Graphs	5	4	1
Trees	1	7	2

Table 2: Distribution of scores by presentation type

arose. This is similar to Mulholland's categorisation (Mulholland, 1994) with the exception that two of Mulholland's three categories: information types and strategies, were collapsed into one on the assumption that information types are implicit in the strategies used by subjects. Additionally, apart from strategies involving the control flow, data flow and goal of the program, which are applicable to a large number of programming activities, the remaining categorical entries were devised specifically for this study.

The strategies and misunderstandings are defined below, followed by a table indicating the number of occurrences of each type per subject. Further discussion of the merits of this type of analysis can be found in Section 2.7.

Strategies

- CF: Reasoning about the control flow of the diagram.
- CORRES: Matching something described in the question, often in the form of hypothesised behaviour) to a particular part in the diagram which represented it (as opposed to, say, starting at the beginning of the diagram and working to that point).
- CV: "Code visualisation" was an attempt by the subjects to mentally visualise the corresponding textual representation of the program code.
- DF: Reasoning about the data flow of the diagram.

- GOAL: Reasoning about the goal of all or part of the program.
- METAA: Rather than using the diagram to reason about the problem, subjects used information from the following sources, or a combination thereof: description of code behaviour, example of code behaviour, code names (which were meaningful in the study and hence led to expectations about the code's behaviour), and information contained in the answers to the multiple choice questions.
- METAC: This is similar to METAA, but applies only to a subset of the diagram, *e.g.* using knowledge of a particular procedure to predict the meaning or value of a particular part of the diagram.

Misunderstandings It should be noted that some of the misunderstandings described are often derived from the incorrect use of a correct strategy, *e.g.* CORRESM, METAAM and METACM.

- CORRESM: A variation on CORRES which involves making an incorrect correspondence between a situation described in the question and the diagram itself, *e.g.* equating part of the execution described in the question with the wrong level in a tree diagram.
- DFM: The subject does not understand the origin of a particular data object, or how it arrived at a particular node in the diagram.
- FMM: Subjects forget the meaning of a particular construct, *e.g.* that a given shape denotes the creation of a data object. This does not in itself describe the action which then followed: either the meaning was correctly guessed, or the concept was ignored (ICM) or misinterpreted (WMM).
- ICM: Subjects do not take a particular diagrammatic construct into account when reasoning with the diagram, which is especially important if it has an effect on the data, *e.g.* a node which takes the first element off a list and returns the rest of the list, as opposed to returning the list intact.
- METACM & METAAM: These are erroneous versions of the METAC and METAA strategies described earlier. Although these strategies are used here, the inferences made on the basis of the various sources are incorrect.
- MISUNDM: This is an underspecified category in the same sense as FMM, as it is unclear what actions followed from it. Although the subject reported that the meaning of a particular object was not understood, he or she did not go on to say whether the object was ignored or its meaning reinterpreted.
- NOCORRESM: The subject is unable to determine, on the basis of a situation given in the question, the corresponding point in the diagram.
- SKIPM: Subjects are at a point in the diagram and skip to another, incorrect, point. This may arise, for example, from an incorrect evaluation of a choice point, which causes them to jump to the wrong node.
- STARTM: Investigation of the diagram starts at an arbitrary point rather than at the start, with the subject ignoring all processing occurring before that point. For example, in a diagram which depicts one procedure calling another recursive procedure, the first procedure, which transforms the input data in some way, is ignored.

- WMM: The wrong meaning is attributed to a construct, or the construct is assumed to have a behaviour which it does not have. This may be due in some cases to short-term memory limitations (*e.g.* the subject forgets the meaning of the construct and attributes an erroneous one) rather than to a deeper misunderstanding. If this is the case, then these errors could possibly be partially eliminated by a change in experimental design.

Table 3 shows the frequency of occurrence of each strategy in total and per group (control or data), while table 4 shows the frequency of occurrence of each type of misunderstanding in total and per group.

Strategy	**Group**		**Total**
	Control Flow	**Data Flow**	
CF	6	4	10
CV	1	4	5
CORRES	14	3	17
DF	4	3	7
GOAL	3	0	3
METAA	5	5	10
METAC	2	0	2
Total	33	19	52

Table 3: Strategies Reported in the Protocol

Misunderstanding	**Group**		**Total**
	Control Flow	**Data Flow**	
CORRESM	4	0	4
DFM	0	1	1
FMM	0	2	2
ICM	3	1	4
METACM	2	0	2
METAAM	0	5	5
MISUNDM	0	4	4
NOCORRESM	4	0	4
SKIPM	2	1	3
STARTM	1	2	3
WMM	2	6	8
Total	18	22	40

Table 4: Misunderstandings Reported in the Protocol

It is interesting to note that many, if not all, of the misunderstandings noted in this study revolved not around comprehension difficulties of program execution per se, but around 1) diagram "navigation", *e.g.* determining where one is in the diagram (CORRESM, NOCORRESM, STARTM), tracing the path which was taken to that point and/or deciding where to go next (DFM, SKIPM) and 2) meaning attribution (FMM, MISUNDM, WMM and, indirectly, ICM).

In terms of breakdown per group, the control flow (CF) group showed more reporting of strategies than the data flow (DF) group: 33 as opposed to 19. At the same time, the reporting of misunderstandings per group was similar (18 CF group, 22 DF group). In the light of the report from many DF subjects that they found the data flow diagrams very confusing and difficult to follow, a speculative hypothesis is that DF subjects were unable to formulate a conscious strategy and follow it through, whereas CF subjects were able to proceed more methodically and report on strategies used.

The results also show that the types of misunderstandings reported varied between groups. While CF subjects reported more difficulty in matching up parts of the question with parts of the representation (4 occurrences each of CORRESM and NOCORRESM in the CF group), DF subjects reported no occurrences of either. However, it would seem that the CF subjects were attempting to use the CORRES strategy more often than the DF subjects (14 reports of CORRES for the CF group; 3 in the DF group), which may account for more erroneous occurrences of it. The DF group reported more difficulty with attributing meaning to concepts (FMM, MISUNDM, WMM) rather than navigational difficulties.

Thus, the qualitative analysis resulted in a classification of the strategies used, and the difficulties involved in diagram navigation and attribution of meaning to the various diagrammatic constructs. In addition, it provides some indication that the difficulties experienced by subjects vary according to the type of information present in the diagram. However, the issue requires further investigation.

Summary. To summarise the results section, it was shown that:

- in terms of the hypothesis that performance will improve if there is a congruency between information content of the representation and the information required by the question, the data did not permit the null hypothesis (that there will be no difference between congruent and incongruent representations) to be rejected;
- the same was true for the representation format hypothesis: although there was a trend for graphs to lead to more accurate responses and a faster reply time, the difference was not significant.
- a qualitative analysis of the types of strategies used and misunderstandings involved allowed subjects' navigational strategies and meaning attribution to be elucidated.

2.7. Discussion

This section looks at the results obtained not only in terms of how they can be accounted for on a theoretical basis, but also considers elements particular to the experiment which may have influenced the results obtained.

The Congruency Hypothesis. As shown above, the congruency hypothesis, that task performance would be improved if the information required by the question was of the same type as that highlighted by the diagram, was not upheld.

It was noted earlier that the congruency hypothesis is similar to Gilmore and Green's match-mismatch conjecture (Gilmore & Green, 1984) and Vessey's concept of cognitive fit (Vessey, 1991) at a high level of abstraction. Indeed, the notion that fitting the representation to the task is beneficial for problem-solving is not a new one, and Norman (1993) nicely illustrates many common examples of this phenomenon. However, although the idea may even seem intuitively obvious, it has little explanatory or predictive power when expressed at this level of abstraction and must therefore be defined in operational terms. Where studies often differ when investigating this hypothesis is in the focus of the match.

Gilmore and Green, in testing the match-mismatch conjecture, used procedural and declarative notions paired with either sequential or circumstantial information. Their hypothesis was that the procedural notation gave better access to sequential information while obscuring circumstantial information, with the opposite holding true for declarative notations, a hypothesis which was confirmed. The match-mismatch conjecture was also tested in conjunction with a comparison between textual and visual representations and again, it was supported (Green & Petre, 1992) for both types of representation.

A study by Vessey (1991) compared the use of tables and graphs for symbolic versus spatial reasoning. After looking at the evidence for the hypothesis that tables support analytic processes while graphs support perceptual processes, she then analysed studies reported in the tables and graphs literature in terms of cognitive fit. Cognitive fit was first determined by independent coders with a very high level of agreement, and support for the theory was found.

In contrast, the current study looked at the congruency between the information content of the diagrammatic representation, which was either data or control flow, and the information required by the task (again, data or control flow), in addition to the effect of the presentation type (tree or graph) on congruency. Thus, although all of the studies looked at the idea of match (or fit, or congruency) between representation and task, the operationalisation of the concept differed in all cases.

Given the results of the present experiment in light of the match-mismatch conjecture and the cognitive fit hypothesis, it might be that their scope is specific to certain types of information and thus cannot account for other information types such as data flow and control flow. This seems unlikely however.

On the other hand, it may be that this type of conjecture may play an important role in a problem solving situation, but that other factors interact to diminish, or even override, its effect. For example, Sinha and Vessey (1992) applied the notion of cognitive fit to the domain of recursion and iteration in programming. Using Lisp and Pascal as the programming languages in their investigation, they found the language itself had a much stronger effect on subjects' performance than did cognitive fit.

Similarly, in the current study, other factors may have contributed to the results obtained: they are described under the headings of previous experience, dominant mental representations, and cognitive load.

Previous experience One explanation for the lack of support for the congruency hypothesis may be that subjects were more familiar with control flow representations, and that this familiarity effect overshadowed the match-mismatch effect. Although the control flow diagrams were not strictly speaking flowcharts, the graphs in particular borrowed heavily from the flowchart tradition. As control flow languages have, in the past, held greater sway than data flow languages, with flowcharts being used to illustrate the workings of the language, it may be that the subjects were more familiar with this type of layout. Investigation of this issue shows that the effect of control flow "superiority" seems to be most marked with respect to graphs (9 out of the 10 questions using control flow graphs were answered correctly, as compared to 4 for control flow trees, 5 for data flow graphs and 5 for data flow trees). Although the data does not allow us to apportion the exact contribution of information type and presentation type (the former is a between subjects factor while the latter is a within subjects factor), there is some indication of where such an effect might lie, and a future experiment using a larger sample size may permit us to do so.

Obviously, this effect, if obtained, may be related to both familiarity with the diagram and/or ease of use. In order to investigate the familiarity issue further in a future experiment, we are considering the use of a variation on the ER taxonomy task (Cox & Brna, 1993) in which control and data flow graphs and trees of the type used in the study would be included in a selection of graphical representations of programs. Subjects could then be asked to rate or group the representations in terms of their familiarity, and the results could be checked for any correlation with subsequent performance measures using the various types of diagrams.

Dominant mental representation A further hypothesis is the extent to which a particular type of representation, in the form of abstractions of computer text mentioned in Section 2, corresponds more closely to the programmer's mental representation of the program (Pennington, 1987). Hence, programs would be represented in memory in a particular way regardless of the form in which they are externally represented. In a study on COBOL and FORTRAN programmers which involved measuring priming effects and comprehension of short programs, Pennington (1987) suggests that programs are represented in memory in the form of abstract program units corresponding to the control flow of the program. However, she also found some language specific results: COBOL programmers performed better on comprehension questions relating to data flow than FORTRAN programmers. Furthermore, COBOL programmers had more difficulty inferring control flow relations.

The idea that the programming language affects the types of relations which are the subject of focus is also supported by Bergantz and Hassell (1991). They found, on the basis of a protocol analysis in terms of frequency and temporal ordering, that program comprehension by Prolog programmers' can be characterised as a two-stage model. The program is first understood in terms of data structures followed by an understanding in functional terms, whereas in a second study by Pennington, again on FORTRAN and COBOL programmers and reported in (Pennington, 1987), program comprehension in procedural, control flow terms is followed by comprehension in functional terms. Therefore, initial understanding at least is influenced by language, with Prolog programmers' understanding beginning with data structure[8] rather than with procedural elements.

If we consider the two different program abstractions, or representations of information, used in this study to be different *notations*, then the implications of the above studies are consistent with the match-mismatch conjecture, and seem to suggest a given notation will affect understanding of the program and its organisation in memory, which will in turn affect task performance. Again, this theory is not upheld by the data obtained in this experiment, in the sense that the two different representations did not positively affect performance on questions requiring the type of data presumed to be highlighted by the representation. Perhaps an explanation lies with other factors.

Cognitive load and the relationship between control and data flow A major factor which may have played a role in the results obtained is the interrelatedness of data and control flow. Data flow and control flow are intertwined: data flow diagrams necessarily comprise control flow information and vice versa. Thus, it may be that, for example, more data flow information was present in the control flow diagrams (in the sense that less inference was needed to access it), while less control flow information was present in the data flow diagrams. There is a need to formalise the definition between data flow and control flow in a way which allows one to explore this issue further. A characterisation of the cognitive load in terms of the procedural steps involved in searching the representations and performing the necessary inference upon the data obtained from them would need to include such points as:

1. the exact information needed to answer the question;
2. how that information is derived;
3. how much of that information is present in the diagram in a "read-off" form;
4. how much needs to be inferred;
5. how many steps are needed in the inference process and what these steps consist of;
6. the information which is needed by each step of the inference process;
7. where that information is derived from (*i.e.* start at step 2 for each piece of information).

These steps could be built into a model which would be executed recursively in order to determine the reasoning and inference processes necessary to answer a particular question with respect to the two types of diagram. This should give a clearer idea of the amount of inference necessary using each type of diagram and as such, determine whether questions said to be congruent are in fact so.

This process may also shed light on a related question: do data flow diagrams really provide the information necessary to answer data flow questions in an easily recoverable way? The control flow

[8] Defined by Bergantz and Hassell as "the type and number of program objects that are transformed during the course of program execution" (Bergantz & Hassell, 1991, p.318).

representations used required following a single arc at any given time. Thus the diagram provided a serial representation of the changes to data objects, which were represented in the nodes. Data flow representations, on the other hand, require following as many arcs as there are data objects, thus necessitating, in the case of more than one data object, a parallel search of the diagram, rather than the serial search afforded by the control flow diagram. This in turn places more of a demand on memory as placekeeping operations must occur for some arcs while search is being directed down others. These operations may make data flow diagrams intrinsically harder to use unless the program consists solely of transformations on a single data object. This hypothesis fits well with Petre and Green's observation that novice and expert behaviour on a graphical programming task was distinguished by the fact that "experts made better use of their fingers, occasionally using all 10 to mark points along a circuit." (Petre & Green, 1993, p.68).

Furthermore, a distinction is made in (Bergantz & Hassell, 1991) between data *flow* relations and data *structure* relations, the former covering the transformations on data objects from input to output over the course of program execution, while the latter concerns the objects themselves in terms of type and structure, as mentioned above. One hypothesis is that the questions asked in this study required information of a data structure type, and that the data flow representations did not give better access to this information than the control flow representations.

Again, a more rigorous analysis would hopefully shed more light on these issues by providing a clearer understanding of the nature of the information required by the question and the processes involved in "extracting" that information from the diagram. However, even if it is possible to determine the amount of information of varying types present in a given diagram, it may be that the nature of both the information and the task requires us to think of terms of *degrees* of match rather than a more categorically defined notion of match versus mismatch, or congruency versus incongruency. In other words, because the information being considered is interrelated, and because the task is such that inference (as opposed to simple read-off) is always necessary, regardless of whether the situation is congruent or incongruent, it may be possible to rank representations in terms of their efficacy rather than simply categorise them.

The Presentation Hypothesis. Although the presentation hypothesis was not upheld, the trend was for graphs to be associated with more correct responses and a quicker response time than trees.

An investigation of why this might be so involves many of the same issues as a further study of the congruency hypothesis: a larger sample size is needed, with four experimental groups (data flow tree, data flow graph, control flow tree, control flow graph), thus allowing the relative contributions of information type and presentation type to be assessed.

Likewise, the issue of familiarity is also relevant here, and hopefully further information could be gained by devising a taxonomy task variation as described in Section 2.7.

Finally, ideas concerning the role of presentation type in such issues as size of search space, role as a memory aid and ease of error recovery mentioned in Section 2 should become clearer through the development of a model of search and inference as described in Section 2.7.

General diagram use. In terms of general observations of diagram use, some of these confirmed earlier findings, particularly the idea of familiarity and the need for training. Green *et al* (1991, 1992) dispute the idea of "graphical superlativism" whereby visual programming languages are "cognitively natural" and hence superior to textual languages, and stress the importance of training and experience in using graphical notations. This was particularly evident for the only subject to obtain correct answers to all questions despite the fact that his score on the pre-test for general recursive skills was one of the lowest. The subject's verbal protocol showed a good understanding of how to navigate through the graph and where to look for relevant information, with no errors of interpretation reported and extensive use of what Green and Petre (1992) term "secondary notation". That subject later reported that he used software engineering graphical representations in his work and felt comfortable reading and writing many different types of models including data flow diagrams and entity relationship diagrams.

Green *et al* also found that "cues imply meaning", in other words, that if a graphical element was visible, it was expected to have some relevance. This phenomenon did not occur in the present study. On the contrary, some subjects seemed to ignore constructs present in the diagram (and necessary in order to correctly answer the question), including any computation which occurred as a result of that construct.

However, this observation may have been due in part to the fact that many subjects used meta-reasoning, basing their answers on expectations stemming from the code name, its described behaviour, examples, or the answers proposed. This may have led them to a less thorough investigation of the graphical representation, using it only as a confirmation of their predicted answer, rather than as a means of obtaining the answer in the first place. This is similar to the situation reported in (Adelson, 1984), in which the availability of program code gave subjects an additional source of information on which to base their representations of a program, which were meant to have been based on an accompanying flowchart describing either the functional or procedural aspects of the program.

Possible solutions to this problem which we are considering include the use of non-meaningful code names and less textual information about the code, or by providing representations which may contain errors and focussing the task on the implications of such errors[9]. We feel that the latter solution would be preferable in that it forces the subject to study the diagram in detail in a potentially realistic setting (*i.e.* part of a programmer's job often involves debugging code written by other programmers, and this should be just as true when the code is visual rather than textual).

Questions of methodology. This section looks at various aspects of the methodology used in the study and how they may have influenced the results or failed to provide the type of results expected.

Screen Recording By recording subjects' mouse movements over the course of the experiment and asking them to comment on the recording after the event, we were hoping to obtain some data on 1) the method used by subjects to search through the diagram and 2) some indication of the inference processes they were engaged in at particular points.

However, the technique proved to be only partially successful, as once subjects saw a particular diagram on the screen, they began to use the mouse in real time to talk the experimenter through their problem solving, rather than following the pre-recorded mouse movements. A possible solution may have been for the experimenter to have access to the controls rather than the subject. However, based on the current experiment, where subjects frequently used the pause button in order to stop the recording and explain something, it may be that visually following the mouse movements, attempting to recall what was happening at that particular time *and* verbalising this to the experimenter may simply be too demanding, in which case, the methodology should be refined for future experiments.

Retrospective Protocol Analysis The preceding paragraph has some implications for the issue of protocol analysis. Rather than being encouraged to verbalise during the experiment, subjects were asked to run through a screen recording of their session after completing the experiment, and one may question the choice of taking verbal protocols after the event rather than during the study. Although this technique may provide an understanding of subjects' strategies and the misunderstandings they held, it is also prone to retrospective rationalisation of behaviour by subjects. On the other hand, concurrent verbalisation places an additional cognitive load on subjects and can significantly change the nature of the study. Both protocol acquisition methods have advantages and disadvantages which merit further study.

[9] Note that this does not preclude testing the congruency hypothesis: *e.g.* questions about the effects of a faulty control flow representation on subsequent data flow could be devised.

3. CONCLUSIONS AND FUTURE WORK

The present study has suggested that the relationship between the task and the information required by the task is not a simple one. It is difficult to imagine that a given representation can simply be deemed to be the "right" one for the task in absence of information on factors such as the features of the representation (what information it contains and in what format), the nature of the task, the match between the two, and the user of the representation, in terms of both experience with the representation and cognitive style preferences.

Although significant results were not obtained in the study, trends were uncovered and possible explanations for these trends were discussed. Future work is planned both in terms of a further experiment and the development of a cognitive model of the processes involved in search and inference over the diagrams. This work will require us to further refine the hypotheses, untangle the variables at work, and hopefully provide greater insight into the processes at work in diagram navigation and reasoning. At present, we have information, in terms of speed and accuracy, on the *outcome* of giving people data flow and control flow diagrams in the form of graphs and trees. We also have some information on the types of strategies they use and the misunderstandings and errors which occur. However, we have little information on the *processes* of navigation and inference preceding the outcome.

The following issues will be the subject of further investigation:

- **Diagram familiarity**: The question of familiarity with regard to information type and presentation type has been considered in Section 2.7 and we hope to incorporate measure for this into further experiments.

- **Cognitive style preferences**: Although familiarity with one type of diagram may facilitate performance using that diagram, some persons may simply perform better using graphical, as opposed to sentential, representations. The idea that "graphical readership" is an acquired skill is not disputed (Petre & Green, 1993), however, there is evidence to support the claim that some individuals are "diagrammatic reasoners" and that their skills in a given domain improve to a significantly greater extent when taught using a graphical teaching method than "non-diagrammatic" reasoners (Cox, Stenning, & Oberlander, 1994; Stenning, Cox, & Oberlander, 1995). Although some studies have used the Paper Folding Test as a measure of visual aptitude (*e.g.* (Cunniff & Taylor, 1987)), we do not feel that this test is appropriate for determining skill in diagrammatic reasoning, and are currently investigating other possible tests.

- **Relative contributions of information content of diagram and presentation type**: Although the desire to reduce numbers required for the pilot study resulted in information content being a between subjects factor and presentation type a within subjects factor, larger sample sizes will be used in the next experiment (as discussed earlier).

- **Differences in search between data flow and control flow diagrams**: It was hypothesised that some of the difficulties observed with data flow diagrams may have arisen from the difficulties of following more than one arc at any given time, with the possible problems of memory overload that this may entail. Mouse movements over the screen are not effective measures of search strategy, particularly since some people use their fingers rather than the mouse and some use neither. The interface to the next experiment is being designed so as to provide finer-grained information on search strategy.

- **Differences in reasoning between diagrams**: We have already discussed possibilities for looking at the reasoning processes involved in using the data and control flow representations to answer data and control flow questions in order to obtain more information on the match between information type and presentation type. This would take the form of a cognitive model described in Section 2.7.

Further experimentation will be carried out in parallel with the development of the cognitive model, so that information obtained from one source can usefully be incorporated into the other.

4. ACKNOWLEDGEMENTS

I would like to thank Paul Brna, Richard Cox and Helen Pain for help with all stages of the experiment and helpful comments on earlier drafts of this paper, and Simon Perkins for extensive input to the representations used in the experiment. Thomas Green and my anonymous reviewers also suggested clarifications and improvements, for which I am grateful.

5. REFERENCES

Adelson, B. (1984). When novices surpass experts: The difficulty of a task may increase with expertise. *Journal of Experimental Psychology: Learning, Memory and Cognition*, *10*, 483–495.

Anderson, J., Pirolli, P., & Farrell, R. (1988). Learning to program recursive functions. In Chi, M., Glaser, R., & Farr, M. (Eds.), *The Nature of Expertise*, pp. 153–183. Lawrence Erlbaum Associates.

Bergantz, D., & Hassell, J. (1991). Information relationships in Prolog programs: how do programmers comprehend functionality?. *International Journal of Man-Machine Studies*, *35*, 313–328.

Brayshaw, M. (in press). Visual models of parallel logic program execution. *Journal of Visual Languages and Computing*.

Cox, R., & Brna, P. (1993). The relationship between external representations and analytical reasoning performance: implications for the design of a learning environment. DAI Research Paper 646, Department of Artificial Intelligence, University of Edinburgh.

Cox, R., & Brna, P. (1995). Supporting the use of external representations in problem solving: the need for flexible learning environments. *Journal of Artificial Intelligence in Education*, *6*(1).

Cox, R., Stenning, K., & Oberlander, J. (1994). Graphical effects in learning logic: reasoning, representation and individual differences. In Ram, A., & Eiselt, K. (Eds.), *Proceedings of the 16th Annual Conference of the Cognitive Science Society*, pp. 237–242. Lawrence Erlbaum & Associates.

Cunniff, N., & Taylor, R. (1987). Graphical vs. textual representation: An empirical study of novices' program comprehension. In Olson, G., Sheppard, S., & Soloway, E. (Eds.), *Empirical Studies of Programmers: Second Workshop*, pp. 114–131 New Jersey. Ablex Publishing Corporation.

Davis, A., & Keller, R. (1982). Data flow program graphs. *IEEE Computer*, *15*, 26–41.

Er, M. (1984). On the complexity of recursion in problem-solving. *International Journal of Man-Machine Studies*, *20*, 537–544.

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

Golin, E. (1991). A method for the specification and parsing of visual languages. Tech. rep. CS-90-19, Brown University, Department of Computer Science.

Green, T., & Petre, M. (1992). When visual programs are harder to read than textual programs. In *ECCE-6: Proceedings of the Sixth European Conference on Cognitive Ergonomics*.

Green, T., Petre, M., & Bellamy, R. (1991). Comprehensibility of visual and textual programs: A test of superlativism against the 'match-mismatch' conjecture. In Koenemann, S., Moher, T., & Robertson, S. (Eds.), *Empirical Studies of Programmers: Fourth Workshop*, pp. 121–146 New Jersey. Ablex Publishing Corporation.

Hils, D. (1992). Visual languages and computing survey: Data flow visual programming languages. *Journal of Visual Languages and Computing*, *3*, 69–101.

Kahney, H. (1989). What do novice programmers know about recursion?. In Soloway, E., & Spohrer, J. (Eds.), *Studying the Novice Programmer*, pp. 209–228. Lawrence Erlbaum Associates.

Kurland, D., & Pea, R. (1983). Children's mental models of recursive logo programs. In *Proceedings of the 5th Annual Conference of the Cognitive Science Society*, pp. 1–5 Rochester, N.Y.

Larkin, J., & Simon, H. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, *11*, 65–99.

Mulholland, P. (1994). The effect of graphical and textual visualisation on the comprehension of Prolog execution by novices: an empirical analysis. In *Collected Papers of the Sixth Workshop of the Psychology of Programming Interest Group*, pp. 18–26.

Norman, D. (1993). *Things that Make us Smart*. Addison Wesley Publishing Co.

Pandey, R., & Burnett, M. (1993). Is it easier to write matrix manipulation programs visually or textually? an empirical study. In *Proceedings of the 1993 IEEE Symposium on Visual Languages*, pp. 344–351.

Pennington, M., & Grabowski, B. (1990). The tasks of programming. In Hoc, J.-M., Green, T., Samurcay, R., & Gilmore, D. (Eds.), *Psychology of Programming*, pp. 45–62. Academic Press.

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

Petre, M., & Green, T. (1992). Requirements of graphical notations for professional users: Electronics CAD systems as a case study. *Le Travail humain*, *55*(1), 47–70.

Petre, M., & Green, T. (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.

Schwarz, B., & Dreyfus, T. (1993). Measuring integration of information in multirepresentational software. *Interactive Learning Environments*, *3*(3), 177–198.

Shu, N. (1988). *Visual Programming*. Van Nostrand Reinhold.

Simon, H. (1969). *The Sciences of the Artificial*. MIT Press, Cambridge, MA.

Sinha, A., & Vessey, I. (1992). Cognitive fit: An empirical study of recursion and iteration. *IEEE Transactions on Software Engineering*, *18*, 368–379.

Stenning, K., Cox, R., & Oberlander, J. (1995). Contrasting the cognitive effects of graphical and sentential logic teaching: reasoning, representation and individual differences. *Language and Cognitive Processes*, *10*.

Tabachneck, H., Leonardo, A., & Simon, H. (1994). How does an expert use a graph? A model of visual and verbal inferencing in economics. In *Proceedings of the 16th Annual Conference of the Cognitive Science Society*. Lawrence Erlbaum & Associates.

Vessey, I. (1991). Cognitive fit: A theory-based analysis of the graphs versus tables literature. *Decision Sciences*, *22*, 219–240.

A CODE FOR PROGRAMS USED IN THE EXPERIMENT

A1. `max` – Finding the maximum number in a list

```
max([H|T],Ans):-
        max(T,H,Ans).

max([],Ans,Ans).
max([H|T],Temp,Ans):-
        H>Temp,
        max(T,H,Ans).
max([H|T],Temp,Ans):-
        max(T,Temp,Ans).
```

A2. `position` – Finding the position of an element in a list

```
position(H,[H|_],1).
position(X,[_|T],N):-
position(X,T,N1),
N is N1+1.
```