BEHAVIOUR & INFORMATION TECHNOLOGY, 1990, VOL. 9, NO. 3, 201–214

Plans, goals and selection rules in the comprehension of computer programs

SIMON P. DAVIES

Department of Computer Studies and Mathematics,
Huddersfield Polytechnic, Queensgate, Huddersfield HD1 3DH, UK

Abstract. The notion of the programming plan has been proposed as a mechanism through which one can explain the nature of expertise in programming. Soloway and Ehrlich (1984) suggest that such expertise is characterized by the existence and use of programming plans. However, studies in other complex problem-solving domains, notably text editing, suggest that expertise is characterized not only by the possession of plan-related structures but also by the development of appropriate selection rules which govern the implementation of plans in appropriate situations (Card *et al.* 1980, Kay and Black 1984, 1986). This paper presents an experimental study which examines the role of programming plans in the context of skill development in programming. The results of this study suggest that plan-based structures cannot be used in isolation to explain novice/expert differences. Indeed, such structures appear to prevail at intermediate levels of skill. The major characteristic of expertise in programming would appear to be strongly related to the development of appropriate selection rules and to so-called program discourse rules. This in turn suggests that current views on the role of plan-based structures in expert programming performance are too limited in their conception to provide an adequate basis for a thorough analysis of the problem-solving activity in the programming domain.

1. Introduction

A number of accounts of human behaviour suggest that performance can be characterized in terms of goals and plans. In particular, when people perform certain activities they devise plans of action and implement them in order to bring about some desired state in the world – that is to say, to achieve a goal. As a result of this a great deal of recent work in cognitive psychology has attempted to make explicit the rules that govern plan-based behaviour and to apply the goal/plan analysis to a range of problem domains. Miller *et al.* (1960) were instrumental in providing a number of central insights into the importance of goals and plans. More recently the goal/plan analysis has been especially important in developing accounts of general problem solving (Newell and Simon 1972), of story understanding (Schank and Abelson 1977) and, during the past few years, in both human–computer interaction (Card *et al.* 1980, 1983, Kay and Black 1984, 1986) and in the psychology of programming (Soloway and Ehrlich 1984, Spohrer *et al.* 1985, Rist 1986, Gilmore and Green 1988).

The computer programming domain, in particular, provides a valuable cache of problems for any goal/plan analysis. The task of programming might best be regarded as a complex communicative process in which abstract structures are used to instruct the computer – in a similar way perhaps to our use of the written word for communication. Soloway and Ehrlich (1984), drawing upon this characterization, suggest that studies of text comprehension can provide a useful foundation for an analysis of programming behaviour. They suggest that expert programmers have

0144-929X/90 $3·00

access to at least two types of knowledge not typically possessed by novices: programming plans – program fragments that represent stereotypical action sequences in programming, and rules of programming discourse – rules that specify the conventions in programming. They suggest that the former correspond directly with the notion of the schema devised by Roger Schank and others in the text comprehension domain (Schank and Abelson 1977, Bower *et al.* 1979) and the latter to the sort of discourse rules which normally operate in conversation. Corresponding to this schemata notion, Soloway and Ehrlich claim that programming plans are generic knowledge structures that provide the basis for the programmer's cognitive representation of the program. Such plans are generated by the rules that govern program discourse and are modified to fit the needs of a particular problem – to achieve a given goal.

Figure 1 shows a goal-plan representation of a Pascal program that reads and sums data until it reaches a termination point and then outputs the average of the data. The top-level goal is, of course, to report an average; however, to reach this goal it is first necessary to accomplish two subgoals: computing the average value and then outputting this value. Such subgoals can, in turn, be further decomposed. The compute average goal can be broken down into three subgoals: computing the sum of the data, computing the number of data items and dividing the sum by the count. Soloway and Ehrlich (1984) suggest that subgoals at this level can be characterized in terms of specific program constructs or plans. Computing the sum of a series of numbers uses what Soloway and his collegues call the running total loop plan. This plan involves initializing the total variable to zero, telling the user to enter a number,

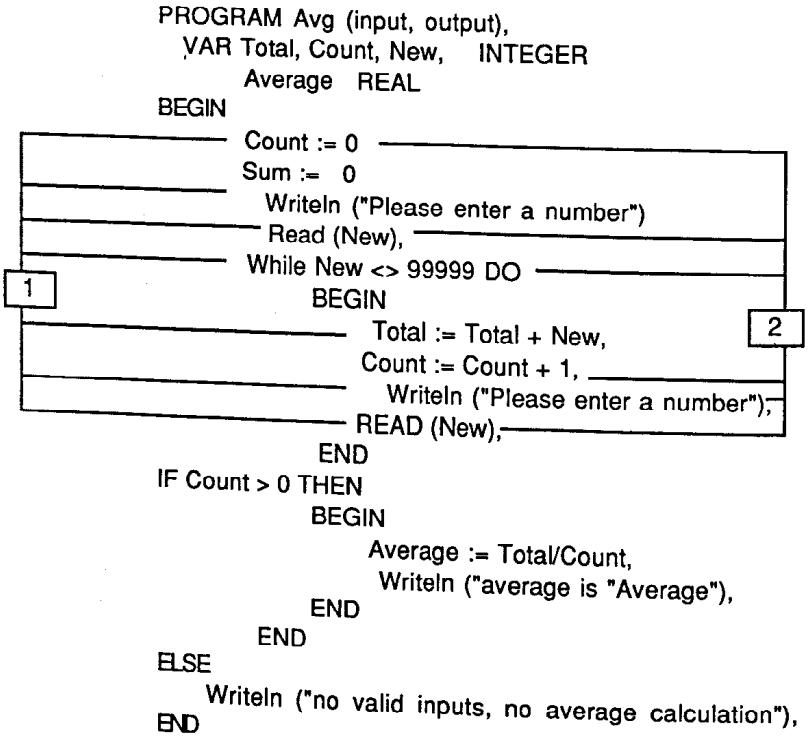

Figure 1. A Pascal program indicating program statements that correspond to a running total loop plan (1) and a counter loop plan (2).

using this to update the total variable and looping until a termination value is encountered. Counting the number of input items uses a counter loop plan, which initializes the count variable, loops like the running total loop plan, but increments the count variable by 1 on each loop rather than updating the total variable. Though conceptually separate, these plans can share many common program statements. Soloway and his colleagues suggest that such plan structures are characteristic of fragments of code that might be found in programs intended to solve a diverse range of problems and correspond to generic knowledge structures that guide the comprehender's interpretations, inferences and expectations.

Soloway and Ehrlich have provided a range of empirical results which seem to verify their goal/plan analysis of the programming activity. In studies of program recall, for example, experts tend to recall program fragments that correspond to plan structures before they recall other elements of the program (Soloway and Ehrlich 1984). This suggests that programmers develop schemas or 'chunks' of knowledge about the domain which represent important functional units in programming. These chunks of knowledge are what Soloway and Ehrlich refer to as programming plans. Expert programmers also have little difficulty in filling in blank lines in programs with appropriate statements when those programs maintain a 'plan-like' structure. When programs do not conform to such a structure, then expert performance does not differ substantially from that of novices in this sort of 'fill in the blank' paradigm (Soloway and Ehrlich 1984).

One criticism of the plan/goal analysis of programming is that it presents a fairly limited view of the programming activity in terms of what we know about the role of plans and goals in other problem-solving domains. Programming plans are proposed as constructs which form the basis for distinctions between novice and expert performance, yet little concern has been directed towards an analysis of the development and use of plan structures and the refinement of goals as programming expertise and knowledge increase. Studies of the development of plan structures and goals in other domains suggest that the plans that underpin expertise develop through a number of identifiable stages. Kay and Black (1984, 1986), for example, have traced the plan acquisition process in a text-editing domain. They suggest that a complex relationship exists between the development of plan structures and increase in expertise. They highlight the importance of the refinement of plan structures and the use of selection rules as expertise develops. In the light of this work, the notion that the primary distinction between the novice and expert programmer is solely based upon the latter's possession of plan-related structures would appear to be an oversimplification. Indeed, experts need not only to possess plan structures but also need to know how to use them appropriately. Kay and Black suggest that during intermediate stages of skill acquisition, plan structures are already well developed but that genuine expertise tends to be exhibited only when appropriate selection rules are developed to guide the implementation of plans.

Kay and Black suggest that skill learning in the text-editing domain progresses through four identifiable stages. The first stage in this description represents the naïve user who has no text-editing experience. They suggest that users bring to the task a range of preconceptions about text-editing terminology which may or may not accord with their interpretations of that terminology as expertise develops. Some evidence for this is presented by Sebrechts *et al.* (1983). In this study it was found that commands such as CENTER and BALANCE or CANCEL and DELETE tend to be grouped together by the novice, even though they bear no similarity in actual function.

The next stage in Kay and Black's description is concerned with the goal of overcoming this prior knowledge bias. They suggest that during this phase (which they call the initial learning phase), users develop conceptual knowledge structures that link specific goals with commands. At this stage users tend to cluster together functionally related commands. For example, INSERT, PUT and REPLACE might be grouped together because these commands are used to accomplish the goal of adding information. Users tend to modify their initial clustering strategy, based upon prior knowledge associations, to one which emphasizes the functional links between commands. Kay and Black suggest that their account of this initial learning phase bears a number of similarities with elements of the GOMS model of Card *et al.* (1980). In particular, they suggest that goal–action relationships are established to form a link between the operators in the GOMS model and particular goals in the task domain. In terms of the GOMS model, goals and operators are acquired first during the initial learning phase, but this occurs at a rather high level of specificity. That is, users tend to understand the general goals involved in text editing and the individual commands that are related to these goals.

The third phase of expertise development is concerned primarily with the formation of plans. Once users have acquired a range of basic editing commands and goals, they learn that a number of commands can be grouped together in terms of the frequency of the use of such commands in accomplishing a particular goal. That is, they combine the actions that were organized separately during the phase of initial learning. Kay and Black suggest that these plans correspond to the methods of the GOMS model. Both Kay and Black and Sebrechts *et al.* provide evidence about the nature of the development of knowledge structures during this third phase. During early stages, users group commands in terms of their functional relationships; as expertise develops this grouping tends to occur with respect to commands that are used in conjunction to accomplish a particular goal. For example, the commands PUT and PICK might be grouped since they are used together when the user wants to move an item of text.

During the final stage in Kay and Black's model, users produce compound plans to accomplish major goals and refine the selection rules that are used to choose among alternative plans in given situations. At this level goals are linked to plans using the conditions in which these compound plans are invoked, whereas in phase 3 goals are linked to simple plans, and during phase 2 merely to actions. Some evidence has been adduced for the evolution of such compound plans. Robertson and Black (1983) found that as experience increased, the time spent pausing between the execution of simple plans decreased, suggesting that users tend to combine these simple plans to form compound plans. Similarly, the time needed to initiate a compound plan decreased as the editing session progressed. For example, in the text editor used by Robertson and Black, to change one word to another three possible plans might be invoked: Type Over and Erase, Type Over or Type Over and Insert. The choice of the optimal plan depends upon the length of the existing word and of that to be inserted. Novice users tend to delete the old word and then insert the new regardless of the relative lengths of the two words. As expertise increases, users begin to use plan structures that are more appropriate to a given situation. For example, if the existing word is shorter than that to be inserted they tend to use an OVERTYPE and INSERT strategy, in the converse situation, OVERTYPE and DELETE are used in conjunction.

The Kay and Black model suggests that a complex relationship exists between expertise and the development of goals and plans in a routine cognitive activity such as text editing. This model also has a number of implications for our understanding of the

role of goals and plans in the programming domain. Previous studies of the programming activity suggest that expertise can be characterized primarily by the possession of plans or plan-related structures and additionally that the existence of such plans can be used to make the distinction between the novice and expert. In the programming domain little or no concern has been directed towards an analysis of the development and refinement of plan structures as expertise increases. The Kay and Black description of skill development suggests a number of key areas of concern for the analysis of problem solving in programming. These can be summarized as follows:

(*a*) Can the mere existence of plans be taken as an indicator of expertise?
(*b*) Are plan structures exhibited at intermediate skill levels?
(*c*) Do programmers develop plan selection rules as their expertise develops?
(*d*) How are plan structures refined as expertise develops?

These issues are here addressed via an experimental study of the programming activity as programming skill increases. Programmers of varying skill levels (novice, intermediate and expert) were presented with a number of Pascal programs, each of which contained several blank lines. In addition, a number of program fragments were presented with each program. The programmer's task was to attempt to state which of the fragments could be used to best complete the program, which might be their second choice and so on. In the first series of programs the associated program fragments represented plan structures and contraventions of plan structures. For example, figure 2 represents a program intended to calculate the square root of a number. The important plan structure in this program might be referred to as a data guard plan (Soloway and Ehrlich 1984). The first program fragment (1) illustrates the correct use of the data guard plan. This plan protects the Sqrt function from trying to take the square root of a negative number. The IF part of the statement carries out this check and makes the number positive if necessary. The second program fragment (2) represents a contravention of the data guard plan. Here the first statement suggests an assignment type initialization (Num: = 0). This gives rise to the expectation of an assignment update (i.e. Num: = Num + 1). However, the data guard plan predicts a read update since using an assignment statement would never result in a negative number – making the data guard plan in this case superfluous. The third program fragment (3) contravenes the plan structure in a more straightforward manner and simply introduces an incorrect test for a negative value (Num > 0) in the last statement.

One might expect, in the light of the studies into the development of expertise that have been reviewed in this paper, that the level of expertise possessed by the programmer would have some effect upon both their choice of the ordering of program fragments and the time taken to make this choice. If the existence of plan structures can provide an indication of the programmers, expertise or, similarly, if such structures facilitate the comprehension of programs, then expert programmers might be expected to choose a fragment that best completes the program which represents or conforms to a plan structure. Indeed, from the results of the Kay and Black studies reviewed in this paper, it might also be expected that programmers of intermediate skill level would make a similar choice. This would confirm the finding that plans exist and are used at both intermediate and expert skill levels. If plan structures are well represented as cognitive schemata, which would be expected in the case of the expert, then the time needed by the programmer to make a choice of appropriate program fragment (conforming to a plan structure) would presumably be less than that needed when such structures are poorly formed (as one might expect in the case of the novice) or when plan structures remain unconsolidated (corresponding to the intermediate level).

The second series of program fragments used in this study represented the correct and violated use of program discourse rules. Program discourse rules, according to Soloway and Ehrlich, specify the conventions in programming. For example, such rules might take the form (following Soloway and Ehrlich 1984):

> If there is a test for a condition then the condition must have the potential of being true.
>
> An IF should be used when a statement body is guaranteed to be executed only once and a WHILE used when a statement body may need to be executed repeatedly.

```
PROGRAM One (input, output),
  VAR Num REAL,
      I   INTEGER,
  BEGIN

          Writeln (Num, Sqrt (Num)),

    END,
  END
```

(1)

```
FOR I = 1 TO 10 DO
   BEGIN
      READ (Num),
      IF Num < 0 THEN Num = -Num,
```

(2)

```
Num = 0,
FOR I = 1 TO 10 DO
   BEGIN
      READ (Num),
      IF Num < 0 THEN Num = -Num,
```

(3)

```
FOR I = 1 TO 10 DO
   BEGIN
      READ (Num),
      IF Num > 0 Then Num = -Num,
```

Figure 2. A program intended to calculate a square root illustrating program fragments corresponding to (1) the correct use of the data guard plan and violations of its use (2) and (3).

```
PROGRAM Three (input, output),
  VAR Max, I, Num, INTEGER,
  BEGIN

    END,
  Writeln (Max),
  END
```

(1)

```
Max = 0
FOR I = 1 TO 10 DO
  BEGIN
    READLN (Num),
    IF Num > Max THEN Max = Num
```

(2)

```
Max = 999999
FOR I = 1 TO 10 DO
  BEGIN
    READLN (Num),
    IF Num < Max THEN Max = Num
```

(3)

```
Max = 0
FOR I = 1 TO 10 DO
  BEGIN
    READLN (Num),
    IF Num = Max THEN Max = Num
```

Figure 3. A program intended to calculate a maximum or a minimum value illustrating program fragments representing the correct use of a program discourse rule (1) and violations of that rule (2) and (3).

To a large extent, rules of programming discourse correspond to the types of selection rules used by experts that have been identified in other studies of skilled performance. Soloway and Ehrlich (1984), for example, claim that discourse rules govern the use of plan structures in programming. That is, they provide an indication of the type of plan that is appropriate at a given point in the program and, in addition, provide some constraints on the formation of plan structures. Figure 3 illustrates the representation of a discourse rule and two violations of the rule in three program fragments. The discourse rule represented in the program is that variable names should reflect their functions. In the example, the program is intended to calculate either a maximum or a

minimum value. The first program fragment (1) represents a procedure that conforms with the program discourse rule. That is, the variable name Max is used in a procedure intended to calculate a maximum value. The second fragment of code (2) represents a violation of this discourse rule. This procedure uses the Max variable in conjunction with the calculation of a minimum value. The last procedure (3) represents a similar violation of this discourse rule, but in this case the program would produce a run-time error if executed.

Again, we might expect experts to exhibit a preference for the program fragment that represents the correct use of a discourse rule, since at this level of skill the selection rules corresponding to the rules of programming discourse will be well developed. Kay and Black suggest that expertise is characterized not only by the possession of plans but also by the use of appropriate selection rules. This also conforms to predictions which stem from the GOMS model of skilled behaviour which places an emphasis on the presence of selection rules as a characteristic of expertise. At intermediate skill levels, it might be expected that programmers, while displaying a preference for plan structures, may not exhibit well-developed selection rules. Hence one might expect that the choice of program fragment used in the completion of a program may not accord with a strategy based upon the use of such rules. Correspondingly, no particular preference for the program fragments that represent discourse rules should be exhibited. This effect should also be apparent in the case of the novice programmer. The time taken to produce an appropriate ordering of program fragments should also provide some evidence about the development and role of program discourse rules at different skill levels. In the case of the expert programmer one might expect discourse rules to be well developed. Hence, the time taken to decide upon an appropriate ordering of rules in a particular circumstance would be less than that required by both the novice and the intermediate programmer. In such cases, we would hypothesize that rules of programming discourse remain undeveloped.

2. Method

2.1. *Subjects*

A total of 45 subjects participated in the experiment. These subjects were categorized according to experience into novice, intermediate and expert programmers. Each of these groups was of equal size. The novice programmers were undergraduates with approximately two months' experience of Pascal. The intermediate group were also undergraduates but this group had completed a nine-month course in Pascal. The expert group were either teachers of Pascal or were employed in industry as programmers. All those in the last group had used the language on a regular basis for more than two years.

2.2. *Procedure*

Subjects were given the programs and program fragments (see figures 2 and 3) in booklet form. These booklets also contained instructions for the completion of the task. Subjects were asked to attempt to complete all the programs (i.e. to choose the most appropriate ordering of program fragments) as quickly as possible. No time limit was imposed on this task and all subjects responded to all the programs. Both the ordering of program fragments and the time taken for each subject to decide upon this ordering for each program were recorded.

2.3. *Materials*

Two sets of programs and program fragments were used in this study. The first set of three programs was associated with program fragments which represented plan structures and contraventions of plan structures. The three program types were as follows:

Program type 1 – A program intended to calculate a square root (see figure 2).
Program type 2 – A program intended to calculate an average.
Program type 3 – A program intended to calculate a maximum or a minimum value (see figure 3).

The plan structures employed (derived from Soloway and Ehrlich 1984) were as follows:

Plan 1 – A guard plan (see figure 2).
Plan 2 – A running total loop plan.
Plan 3 – A search plan.

Plan 1 was associated with program 1, plan 2 with program 2 and plan 3 with program 3. Each program was presented with three program fragments (as in figures 2 and 3). One program fragment represented the correct use of the plan and the second and third fragments a contravention of the programs plan structure. The ordering of program fragments presented with each program was randomized, as was the order in which program types occurred.

The second series of programs consisted of the same program types (program types 1–3) and a number of associated program fragments. These fragments represented program discourse rules and violations of these rules (see figure 3). The three discourse rules used (derived from Soloway and Ehrlich 1984) were as follows:

Discourse rule 1 – If testing for a condition then the condition must have the potential of being true.
Discourse rule 2 – Do not include statements in the program which will not be used.
Discourse rule 3 – Variable names should reflect function.

Each of the three program types was presented with three associated program fragments. One of these program fragments represented the correct use of the program discourse rule, a second represented a violation of the rule but resulted in an executable program, and the third violated the discourse rule but resulted in a error-prone program (see figure 3). In the same manner as above, discourse rule 1 was associated with program 1, discourse rule 2 with program 2 and discourse rule 3 with program 3.

3. Results

Figures 4–7 illustrate the results of this study. Figure 4 shows the programmers' first choice of program fragment when completing a program. These fragments correspond either to a plan structure or to plan structure violations. As can be seen, both intermediate and expert programmers choose, in the main, to complete the program with the fragment that corresponds to the correct use of a plan structure. Conversely, novices do not exhibit a preference for plan structures over program fragments representing violations of plan structure. These effects were statistically significant.

There was an overall effect of skill level ($F_{2,28} = 13{\cdot}64$, $p < 0{\cdot}001$) and of fragment type (plan/plan violation) ($F_{2,28} = 10{\cdot}62$, $p < 0{\cdot}001$). In addition, the interaction

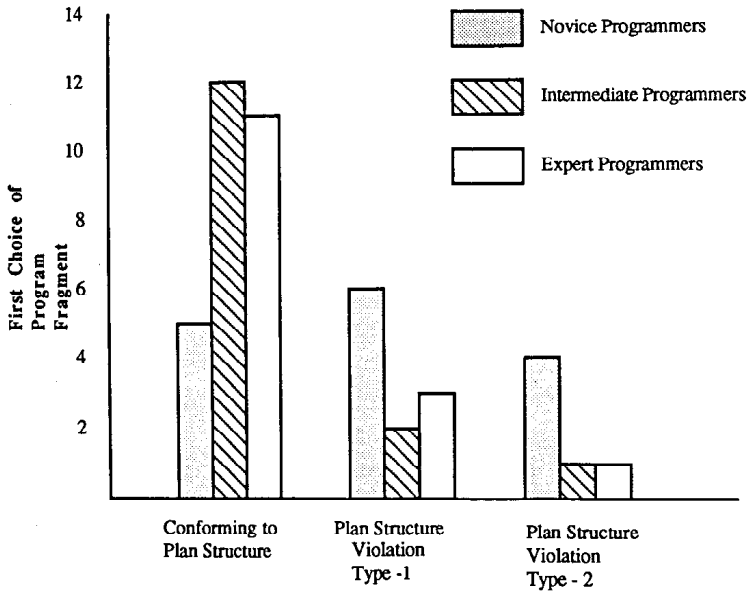

Figure 4. The frequency with which a program fragment (corresponding to a plan structure and violations of plan structure) was chosen as best completing a program by novice, intermediate and expert programmers.

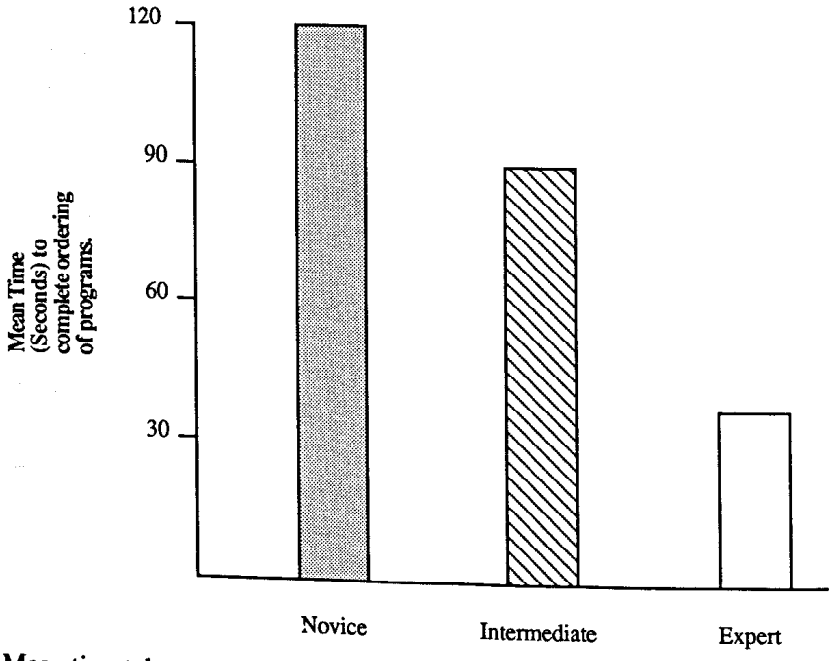

Figure 5. Mean time taken to order program fragments (corresponding to a plan structure and to violations of plan structure) by novice, intermediate and expert programmers.

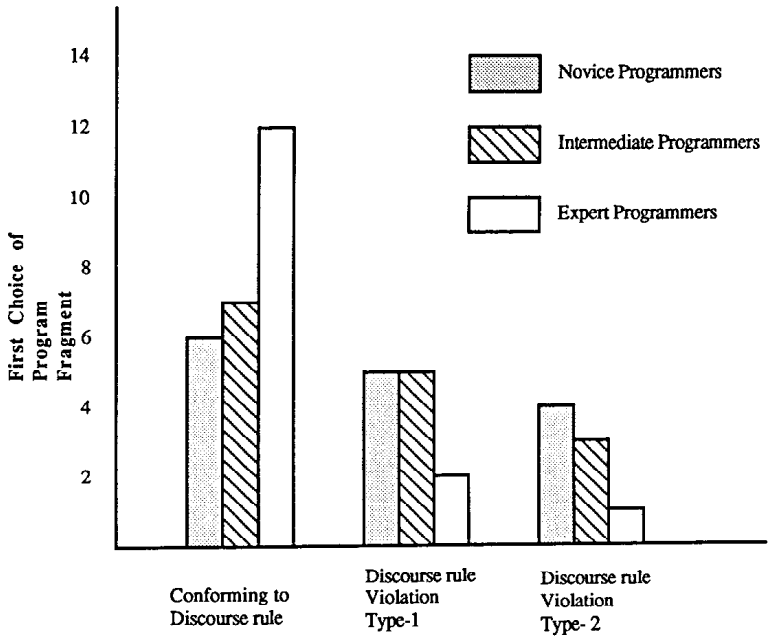

Figure 6. The frequency with which a program fragment (corresponding to a discourse rule and to violations of a discourse rule) was chosen as best completing a program by novice, intermediate and expert programmers.

between skill level and fragment type (plan/plan violation) was also significant ($F_{4,56}=5{\cdot}92$, $p<0{\cdot}001$). Multiple *post hoc* comparisons were conducted with the Newman-Keuls test, and a significance level of $p<0{\cdot}01$ was adopted for all such tests. This procedure indicated that both expert and intermediate programmers choose the program fragments representing plan structures more frequently than novices. Correspondingly, novices tended to choose fragments representing plan structure violations more frequently than both intermediate and expert programmers. None of the other contrasts between means was significant. Hence, novices choose fragments representing plan structures with approximately the same frequency as they choose fragments representing plan structure violations and intermediates choose to use the correct plan fragments with about the same frequency as experts.

Figure 5 shows the total time taken by novice, intermediate and expert groups to make a choice of program fragments and to order them appropriately. Novices take the greatest time to order the program fragments, followed by the intermediate group and then by experts. These differences are statistically significant. Novices take significantly longer than those in the intermediate group to order program fragments (t-test, $p<0{\cdot}05$, two-tailed), and the latter group also take somewhat longer than experts (t-test, $p<0{\cdot}05$, two-tailed).

Figure 6 shows the mean number of occasions in which program fragments, corresponding to the correct and violated use of a program discourse rule, are chosen first in the ordering of program fragments by novice, intermediate and expert programmers. Here, when asked to complete a program, the expert group tend to choose first the program fragment which represents the correct use of a program discourse rule. Those in the intermediate group tend to make this choice less frequently. Novices exhibit no particular preference for the correct use of a discourse rule over program fragments representing contraventions of the rule. These effects were statistically significant.

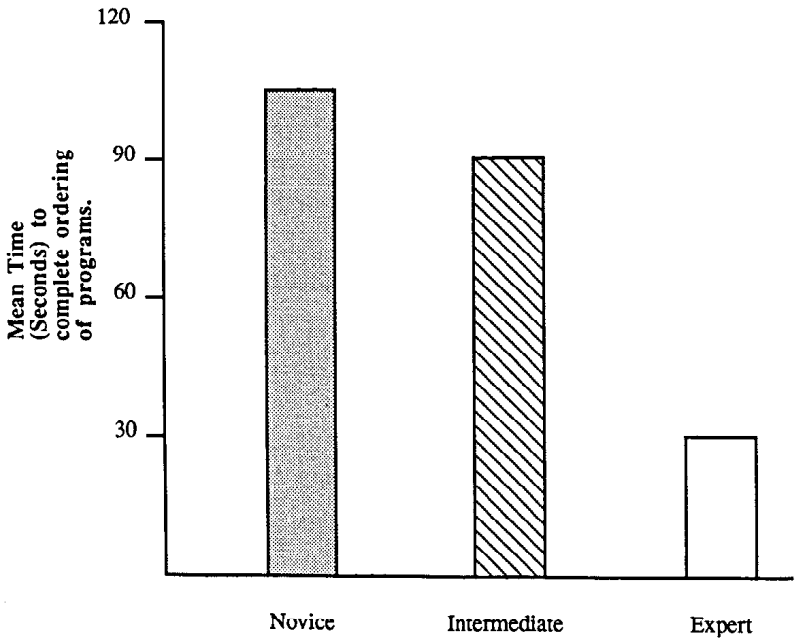

Figure 7. Mean time taken to order program fragments (corresponding to a discourse rule and to violations of a discourse rule) by novice, intermediate and expert programmers.

There was an overall effect of skill level ($F_{2,28} = 8{\cdot}63, p < 0{\cdot}001$) and of fragment type (discourse rule/discourse rule violation) ($F_{2,28} = 19{\cdot}54$, $p < 0{\cdot}001$). The interaction between skill level and fragment type (discourse rule/discourse rule violation) was also significant ($F_{4,56} = 11{\cdot}46$, $p < 0{\cdot}001$). Once again, multiple *post hoc* comparisons were conducted using the Newman-Keuls test, and a significance level of $p < 0{\cdot}01$ was adopted. This procedure indicated that experts choose the program fragment representing the correct use of a program discourse rule more frequently than both the novice group and the intermediate group. Comparing the choice of program fragments representing the correct and violated use of a program discourse rule suggests that experts tend to choose the program fragment that represents the correct use of such a rule more frequently than a fragment representing a contravention of the rule. The same is true of the intermediate group. Conversely, novices show no preference for fragments representing the correct use of a discourse rule compared with those representing the contravention of such a rule.

Figure 7 shows the total time taken by novice, intermediate and expert groups in completing the ordering of program fragments representing discourse rules and discourse rule violations. Here, experts tend to complete the task faster than both intermediate and novice groups. This is confirmed by further statistical analysis. Experts order program fragments faster than both those in the intermediate group (t-test, $p < 0{\cdot}05$, two-tailed), and those in the novice group (t-test, $p < 0{\cdot}05$, two-tailed). Subjects in the intermediate group, in turn, appear to complete this ordering task slightly more quickly than those in the novice group, but this difference was not significant.

4. Discussion

The results of this study suggest that the relationship between programming skills and the formation and utilization of plans and selection rules in programming is by no means straightforward. Previous work in this area suggests that expertise is

characterized primarily by the possession of programming plans and rules of programming discourse (Soloway and Ehrlich 1984). This work, however, fails to examine the way in which such plans might be formed and the nature of their use as skill in programming develops. The results of the experiment reported above suggest that programming plans exist at both expert and intermediate skill levels in programming. At both levels such plans provide a basis for the comprehension of programs. Both expert and intermediate groups are equally likely to choose a program fragment corresponding to a plan structure when completing a program. Novices, in contrast, appear to exhibit no particular preference for those program fragments representing plan structures over those representing plan structure violations.

One interesting difference in performance that emerged between the expert and intermediate groups was that reflected in the time taken to complete the ordering of program fragments corresponding to plan structures and to plan structure violations. The intermediate group took much longer than the expert group to produce such an ordering. This may suggest either (*a*) that while both intermediate and expert groups utilize plan-based structures, in the case of the former these plans remain unconsolidated or (*b*) that members of the intermediate group are, for some reason, unable to easily access or activate these plans.

These findings would accord with results obtained from studies of the development of skilled performance in other problem-solving domains. For example, Kay and Black (1984) suggest that the existence of plan structures in text editing is characteristic of both intermediate and expert performance, but that the rules of selection governing the use of appropriate plan structures only develop at higher levels of expertise. Such expertise would appear, in addition, to be characterized by the development of so-called compound plans. Indirect evidence for this is adduced by Kay and Black from studies which suggest that as expertise develops the time taken to formulate and implement plans decreases. Another possible interpretation of this result might be to suggest that experts automate some simple generic subcomponents of the programming task (these may correspond to plan structures). Empirical studies of this knowledge compilation process have been reported for general problem-solving tasks (Anderson 1982) and within the more specific context of programming (Anderson 1987, Wiedenbeck 1985).

The results of the experiment reported in this paper provide some support for the contention that expertise in programming cannot be explained merely by alluding to the notion of the so-called programming plan. Such plans are used by both intermediate and expert programmers. The important distinction between these groups would appear to be based more strongly upon the use and deployment of appropriate selection rules as expertise develops.

This distinction appears to be reflected in the results stemming from an examination of the role of program discourse rules in skill development in programming. Expert programmers tend to make the most use of program discourse rules. Their use of such rules differs significantly from that of intermediate programmers who, when completing a program, tend to exhibit no particular preference for the program fragment representing the correct use of a discourse rule over those violating such rules. This would suggest that an important characteristic of expertise is related to both the possession and the use of program discourse rules. Examining the time taken to order program fragments which correspond to discourse rules and to violations of these rules indicates that intermediate performance differs little from that of novices. Experts, on the other hand, perform the task a great deal

faster than both novice and intermediate groups. This again supports the view that such discourse rules are an important feature of expert performance.

Existing plan/goal analyses of programmer behaviour provide only a limited insight into some of the underlying features of this important problem-solving activity. Studies in other domains, most notably that of text editing, suggest the need to examine in more detail not only the role of plans but also the nature of the development and refinement of such plans as expertise increases and, in addition, the central role played by selection rules in expert performance. This paper has attempted to highlight the correspondences that exist between plan/goal analyses of text editing and those which appertain in programming. Strong similarities have emerged between these domains. Models of problem solving such as that proposed by Kay and Black for text editing provide a valuable basis for an analysis of programming. Extending the scope of such models to account for performance differences in programming has highlighted a number of deficiencies in the current plan/goal analysis of problem solving within this domain. The present study has attempted to address some of these deficiencies and by doing so to suggest ways in which the plan/goal analysis of programming might be extended. The central theme of the paper – that programming plans alone do not provide an adequate basis for a full account of expert problem solving in programming – is supported and the plan/goal analysis of programming is extended to reflect the central role played by selection rules in expert performance.

References

ANDERSON, J. R. 1982, Acquisition of cognitive skill. *Psychological Review*, **89** (4), 369–406.

ANDERSON, J. R. 1987, Skill acquisition: compilation of weak-method problem solutions. *Psychological Review*, **94** (2), 192–210.

BOWER, G. H., BLACK, J. B., and TURNER, T. J. 1979, Scripts in memory for text. *Cognitive Psychology*, **11**, 177–220.

CARD, S. K., MORAN, T. P., and NEWELL, A. 1980, Computer text-editing: an information processing analysis of a routine cognitive skill. *Cognitive Psychology*, **12**, 32–74.

CARD, S. K., MORAN, T. P., and NEWELL, A. 1983, *The Psychology of Human–Computer Interaction* (Hillsdale, NJ: Lawrence Erlbaum Associates).

GILMORE, D. J., and GREEN, T. R. G. 1988, Programming plans and programming expertise. *Quarterly Journal of Experimental Psychology*, **40A** (3), 423–442.

KAY, D. S., and BLACK, J. B. 1984, The changes in knowledge representations of computer systems with expertise. *Proceedings of the Human Factors Society, 28th Annual Meeting, Santa Monica, USA.*

KAY, D. S., and BLACK, J. B. 1986, Knowledge transformations during acquisition of computer expertise. CCT Report 86-2, Teachers College, Columbia University, New York.

MILLER, G. A., GALANTER, G., and PRIBRAM, K. H. 1960, *Plans and the Structure of Behaviour* (New York: Holt, Rinehart & Winston).

NEWELL, A., and SIMON, H. A. 1972, *Human Problem Solving* (Englewood Cliffs, NJ: Prentice-Hall).

RIST, R. 1986, Plans in programming: definition, demonstration and development. *Empirical Studies of Programmers*, ed. E. Soloway and S. Iyengar (Norwood, NJ: Ablex).

ROBERTSON, S. P., and BLACK, J. B. 1983, Planning in text editing behavior. *Human Factors in Computing Systems*, ed. A. Jonda (New York: North-Holland).

SCHANK, R., and ABELSON, R. P. 1977, *Scripts, Plans, Goals and Understanding* (Hillsdale, NJ: Lawrence Erlbaum Associates).

SEBRECHTS, M. M., BLACK, J. B., GALAMBOS, J. A., WAGNER, R. K., DECK, J. A., and WIKLER, E. A. 1983, The effects of diagrams on learning to use a system. Learning and Using Systems Project Report 2, Yale University, New Haven.

SOLOWAY, E., and EHRLICH, K. 1984, Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*, **SE-10**, 595–609.

SPOHRER, J. C., SOLOWAY, E., and POPE, E. 1985, A goal-plan analysis of buggy Pascal programs. *Human–Computer Interaction*, **1**, 163–207.

WIEDENBECK, S. 1985, Novice/expert differences in programming skills. *International Journal of Man–Machine Studies*, **23**, 383–390.