Journal of Experimental Psychology:
Learning, Memory, and Cognition
1984, Vol. 10, No. 3, 483-495

When Novices Surpass Experts: The Difficulty of a Task May Increase With Expertise

Beth Adelson
Harvard University and Cognitive Science Program, Yale University

A detailed look at the representations constructed by novice and expert computer programmers is presented. The issue is addressed by looking at the interaction between the representation naturally formed by programmers at each level of expertise and an experimentally induced abstract or concrete "mental set." The data suggest that, in the absence of an experimentally provided set, experts form *abstract* representations (defined here as what a program does), whereas novices form *concrete* representations (defined here as how a program functions). The data also suggest that appropriate sets can aid each group to form the representation not natural to them; however, these representations are not as stable as the preferred ones. The generality of the findings and the utility of the experts' representation of a task is discussed both in relation to computer programming and to problem solving in other domains.

In the semantically rich domains of skilled problem solving (Bhaskar & Simon, 1977), the difference between experts and nonexperts is both qualitative and quantitative. Not only do experts perform better than novices on quantitative measures of skill but experimental manipulations also uncover qualitative differences in the representations and strategies used by experts. Repeatedly, we find that the working representations of experts are abstract conceptualizations of the original problem statement, whereas those of novices are less abstract and focus more on surface features of the problem. For example, in a recent experiment, Adelson (1981c) found that expert programmers used abstract, conceptually based representations when attempting to recall programming material, whereas novices used syntactically based representations. Using a multitrial free-recall procedure, Adelson asked novice and expert programmers to recall a set of 16 lines of programming code that had been presented in random order. Although the subjects had not been told that the 16 lines could be organized either conceptually into three programs or syntactically into five categories according to the control words that they contained, analyses of the order of recall for each group showed that the experts had clustered the lines into complete programs, and the novices had clustered the lines according to syntactic categories.

The classic result on the abstract nature of the representations of experts was obtained by Chase and Simon (1973). They replicated de Groot's (1965) findings in which Master chess players reconstructed with greater than 90% accuracy midgame boards that they had seen for only 5 s. They then went on to isolate and to characterize the chess Masters' recall clusters and found that clusters frequently consisted of chess pieces that formed attack or defense configurations. This observation suggests an abstract representation in which individual chess pieces are seen as integral parts of larger, meaningful units. Looking at the recall clusters of Master Go players, Reitman (1976) also found abstract representations that were based on the attack and defense relationships in the game board. McKeithen, Reitman, Rueter, and Hirtle (1981) found that intermediate programmers cluster the words of a programming language by concept, whereas beginners cluster the same words alphabetically. Schneiderman's (1977) finding that the recall distortions of skilled computer programmers

This research was supported by National Science Foundation Grant BNS-79-12418 to Stephen M. Kosslyn.

Thanks to Elliot Soloway, R. Duncan Luce, Miriam W. Schustack, Edward E. Smith, Roger Brown, Stephen M. Kosslyn, and W. K. Estes for their advice and guidance.

Requests for reprints should be sent to Beth Adelson, who is now at the Cognitive Science Program, Yale University, New Haven, Connecticut 06520.

preserve the concepts but not the specific form of previously seen material also suggests an abstract representation.

Chi, Glaser, and Rees (1981) have drawn inferences about the schemata of novice and expert physicists from the results of conceptual sorting tasks and verbal protocols. They suggested that the schema of the novice represents the surface features of the problem, whereas the schema of the expert represents the abstract physical principles involved plus conditions that specify when to apply the principles.

Lewis (1981) examined the solutions of experts and novices in algebra problems. He found that experts often restructure the terms in the original problem, but novices never do. The kind of restructuring that Lewis found suggests an abstraction of the elements of the problem. For example, complex subexpressions are replaced by single, temporary variables that more easily allow the appropriate manipulations.

Taken together, the above findings seem to support the suggestion that experts form abstract, conceptual representations of problems, but novices form representations that tend to retain the surface elements of the problem. The research presented in this article gives us a more detailed look at the representations of the novice and expert problem solvers by characterizing some of the properties of the abstract and concrete representations formed during the comprehension of computer programs. Given the kind of information needed to comprehend programs and given the findings that experts form abstract representations, it seemed plausible that in comprehending programs, experts might form abstract, conceptual representations that are concerned only with what the program does. (This may be thought of as a representation that includes information about the output or result of the program. This point is elaborated on shortly.) It also seemed plausible that novices might form concrete representations, specifically concerned with how the program functions and not with abstract descriptions of what it does. (This may be thought of as the algorithm or specific method used to achieve the result.)

In the previous paragraph and throughout this article, certain aspects of representations are considered abstract and certain aspects are considered concrete. The motivations for these distinctions are central to this article, and so deserve further explanation. The distinctions are explained first, followed by an account of why the particular aspects are abstract or concrete.

Several disciplines have found it useful to distinguish between the description of *what* something does and *how* it is done. Ryle (1949) phrased this as the distinction between knowing *how* and knowing *that*. For example, without knowing how to make it do so, we can know that a Stradivarius makes a beautiful tone. In this article, which focuses on representations for computer programs, the distinction is stated as the difference between *what* a program does and *how* it does it. This seems equivalent to Winograd's (1974) distinction between the procedural and declarative knowledge of a program.

What the program does is equivalent to what is technically called the *output specification;* it refers to the result that the procedure accomplishes. How the program does it is called the *algorithm* and refers to the method used to achieve the result. An explanation follows of why this distinction is appropriate here.

The programs used in this experiment were written in polymorphic programming language (PPL). PPL was designed to be general to Algol-like, high-level, structured programming languages. These languages form the majority of languages currently used in both educational and industrial settings. In the Algol-like family of programming languages, there are relatively few high-level constructs available to the programmer. What this means is that the programmer ultimately has to specify all of the steps to be taken to produce any given result. The contrasting case is a language in which certain computational procedures are built in, and the programmer can cause the program to perform them during program execution just by using the name of the procedure, rather than by specifying what must be done step by step. For example, in PPL if a programmer wanted to create an $i \times j$ matrix whose cell entries were the average of each of the corresponding cells in a set of n other $i \times j$ matrices, he or she eventually would have to state explicitly that a set of cells should be summed and then divided by n and also that this procedure should be repeated for each of the $i \times j$ cells. On the other hand, in the non-

Algol language APL, matrices of numbers can be averaged just by saying the APL name of the "average these matrices" procedure. The point here is that for the most frequently used programming languages, the distinction can be made between what is to be done (e.g., averaging a set of matrices) and how that is to be accomplished (e.g., by repeatedly summing and dividing). This distinction can be made because the programmer can describe (either internally or to others) what is to be done, yet he or she also has to state in the program the mechanism for doing it. These distinctions would not necessarily hold for some of the very high-level, state-of-the-art languages that are now being developed (as in the preceding APL example), however, they do hold both for the programming environments in which today's programmers have acquired their skills and for the environment in which they currently practice them.

Given that the above distinctions are valid, let us look at how they map onto the abstract and concrete aspects of the programmer's knowledge. Describing only what a program does (in terms of result accomplished) is abstract in the sense that a single result can be accomplished in a variety of ways (e.g., in order to alphabetize a set of references, I can use Merge sort, Bubble sort, or Shell sort, each of which uses a very different method, Adelson, 1981a, 1981b, but produces exactly the same result). In comparison, describing how a program functions (in terms of the method that it employs) is more specific and concrete. These distinctions should be thought of as points along the dimension of abstraction, rather than as absolute end points.

Experiment 1 tests the hypothesis that during program comprehension, novices form representations that include information about how programs function and exclude information about what programs do, whereas experts form representations which include information about what programs do and exclude information about how they function. If this hypothesis is supported, we gain information about the nature of the chunks that we repeatedly find the expert using. That is, we not only see that these abstract chunks are being used but we also see what information is regularly included in them as useful and what is left behind as irrelevant detail. In addition to giving us information about the properties of the representation that is naturally formed by novice and expert programmers when comprehending a program, the methodology used also gives us information on two other issues: First, how does the representation used by each group affect their performance? That is, what tasks are facilitated and what tasks are hindered based on the information that is included in the representations that are formed? And second, how flexible can the novices and the experts be? Can they form representations that are not initially natural to them, and if so, how effectively can they then use the information contained in these representations?

Experiment 1

In order to test the hypothesis that during program comprehension an expert represents a program abstractly, whereas a novice represents it in more concrete terms, groups of novice and expert programmers were given tasks in which they had to form and to use both types of representations. It was then possible to look at the performance of each group in each situation and to make inferences about their accustomed way of representing and using programming information.

Method

Subjects

The subjects formed two groups, Novices and Experts. The Novice group consisted of 18 Harvard undergraduates who had completed an introductory course in computer programming. The Expert group consisted of 18 teaching fellows for that same course. All subjects were paid volunteers.

Stimuli

The stimulus set consisted of eight PPL programs with two types of flowcharts and two types of questions for each of the eight programs. Both the contents and the syntax of the programs were chosen so that they would be familiar to both groups. The length and complexity of the programs were approximately equivalent to programs assigned as 2-week problem sets for the introductory course so that they would be manageable to both groups.

The flowcharts were of two types. One type described the output resulting from the program, and the other described how the program functioned. The flowcharts that described only what the program did (in terms of result accomplished) are referred to as the *abstract* flowcharts. As stated earlier, this is because they describe what result

is achieved without specifying what method is being used to achieve it. On the other hand, the flowcharts that described how the programs function are referred to as the *concrete* flowcharts, because they represent only the step-by-step mechanics of the program without providing any general descriptive information about what results are derived from the series of steps. The questions consisted of abstract questions that asked something about what each program did and concrete questions that asked something about how each program functioned. Figure 1 represents one of the abstract flowcharts used in Experiment 1. It explains that the program creates a two-dimensional array that is 40 characters horizontally and 20 characters vertically and that the array has its boundaries marked with a border of *b*s.

Figure 2 represents the concrete flowchart for the same program. It tells the subject that the program first creates a two-dimensional array. It then fills in first the top and then the bottom borders and then goes back and fills in the left and right borders. Although the two sample flowcharts are an extreme example of the difference in length between the abstract and the concrete flowcharts, all the concrete flowcharts were longer than the abstract flowcharts.

Figure 3 represents the program that was described in the preceding two flowcharts. Although comments are included here to guide the reader, subjects saw an uncommented version.

The following are the abstract and concrete questions for the program presented here.

Abstract question (what the program does):
Is the field wider than it is long?
Answer: Yes. (Width = 40. Length = 20.)
Concrete question (how the program works):
Which border of the field is filled in first?
Answer: Top.

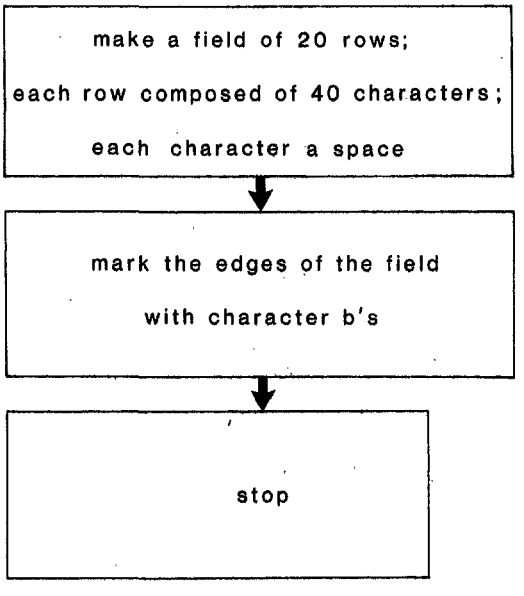

Figure 1. Example of an abstract flowchart. (This describes what the program does.)

For each program, subjects saw one question or the other (depending on the level of abstraction and set for that condition). The answer is presented here for the reader; of course it was not presented during the experiment. Although the lengths of correct answers varied somewhat across items, all tended to be short and of equal length for both abstract and concrete questions.

Design

The level of abstraction of the flowchart was crossed with the level of abstraction of the question to form four conditions for 12 subjects at each level of expertise: two appropriate set conditions, in which the level of abstraction of the flowchart and the level of abstraction of the question match, and two inappropriate set conditions, in which the level of abstraction of the flowchart and the level of abstraction of the question do not match. In the two appropriate set conditions, subjects saw either an abstract flowchart and then an abstract question or a concrete flowchart and then a concrete question. In the two inappropriate set conditions, subjects saw either an abstract flowchart and then a concrete question or a concrete flowchart and then an abstract question.

The two dependent variables are comprehension time (the time it took the subject to say that he or she understood the flowchart well enough to go on and study the program with its question) and error rates on the questions. Question time, the interval between seeing the program along with the question and being able to write down the answer, was also recorded and correlated with error rates to investigate the presence of speed–accuracy trade-offs. (An alternative measure of question time is the interval from when the subject first saw the program until he or she completed writing an answer; however, this longer interval includes a measure of the subject's writing speed as well as processing speed. Therefore, this interval was thought to be a less accurate measure of processing time and was not used.)

Each subject saw flowcharts at only one level of abstraction; however, all subjects were asked questions that were both appropriate and inappropriate to the level of their flowcharts (four of each). Each subject saw only one question for each of the eight programs. Programs, questions, and question types were counterbalanced across subjects and across conditions. The ordering of appropriate and inappropriate questions was random. Subjects within a given level of expertise were assigned randomly to a flowchart level of abstraction.

Procedure

In order to encourage subjects to form a representation at a given level of abstraction, subjects were first shown a flowchart of a program at that level of abstraction, and comprehension time was recorded. The flowcharts served to form a mental set for the subjects. They were then shown the actual program along with a question about the program, and question time was recorded. This procedure was repeated for each of the eight programs. A stopwatch was used to record times, and subjects verbally indicated their readiness to respond. (Because the program and the flowchart were presented together, the subjects could read the question before studying the program. This point is discussed later.)

A code only condition was included in which 6 Expert and 6 Novice subjects saw the program itself immediately and then saw the same program again with the question.

Results and Discussion

Comprehension Task

Presented in Figure 4 are the results of the comprehension time measure (in minutes and seconds) for both the Novice and the Expert groups. Because this initial measure showed no significant difference for the two groups, it is unlikely that the performance differences we see on the question time measure result from differences in the amount of study time each group had ($F < 1$). The code by itself took the longest to comprehend. The concrete flowcharts were comprehended more quickly, and the abstract flowcharts were comprehended most quickly of all. In minutes and seconds, Novices = 1:39, Experts = 1:35; Novices = 1:29, Experts = 1:22; Novices = 0:20, Experts = 0:19; for code only, concrete flowcharts and abstract flowcharts, respectively. The three conditions showed a significant linear trend; $F(1, 24) = 22.15$, $p < 01$. It seems that the flowcharts at both levels of abstraction are reducing comprehension time. (Note that the analyses for comprehension time, Experiment 1, and task performance,

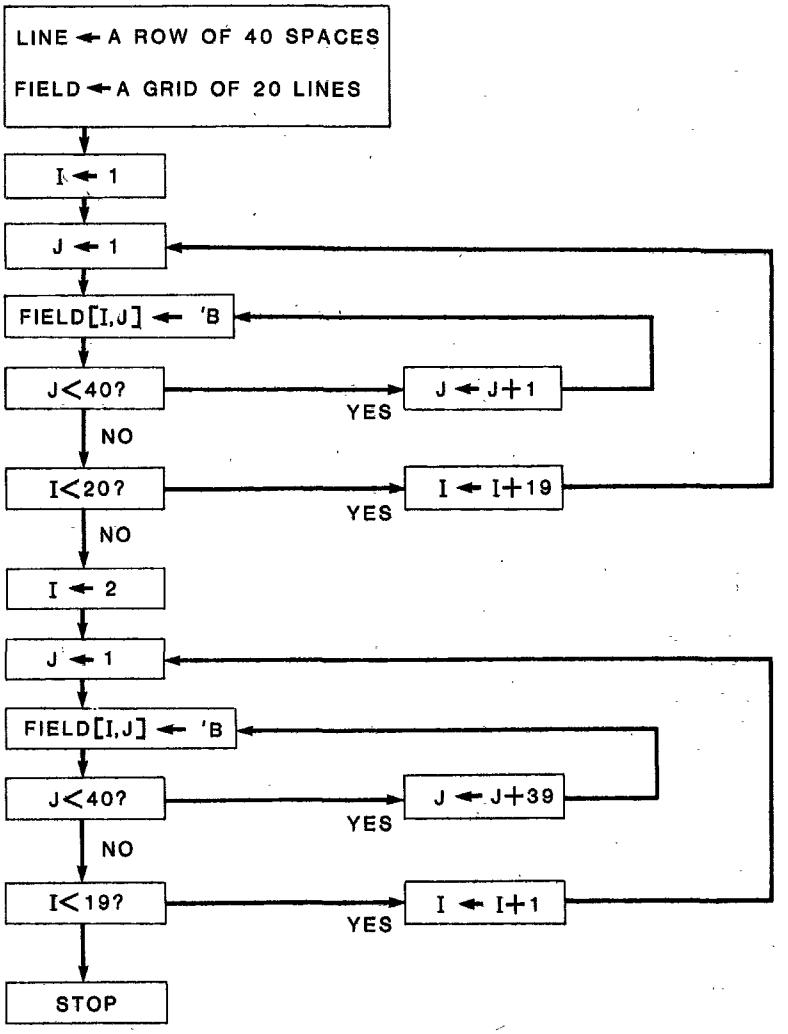

Figure 2. Example of a concrete flowchart. (This describes how the program functions.)

```
PROGRAM

$MAKEFIELD; I;J
[1] ...DATA DEFINITIONS IN THE CURRENT ENVIRONMENT
[2] ...$GRID=[1: ] ROW
[3] ...$ROW=[1: ] CHAR
[4] LINE←MAKE (ROW,40,')          ...CREATE THE FIELD
[5] FIELD←MAKE (GRID,20,LINE)
[6] FOR I←1:19:20 DO %          ...FILL IN TOP & BOTTOM
    (FOR J←1:40 DO FIELD [I,J]←'B)          ...BORDER
[7] FOR I←2:19 DO %
    (FOR J←1:39:40 DO FIELD [I,J]←'B) ...FILL IN SIDES
```

Figure 3. Program described by the flowcharts shown in Figures 1 and 2.

Experiment 2, include the code only conditions. The analyses of response time and error rates for questions do not include the code only conditions because the appropriateness of set factor was not present in these conditions.)

Responses to Questions

The main effects and two-way interactions are presented here followed by a detailed look at the results of the appropriate and inappropriate set conditions. The effects of expertise, $F(1, 16) = 2.67$, $p > .10$; abstraction of set ($F < 1$); and appropriateness of set ($F < 1$) were not significant. The Expertise × Abstraction of Set interaction, $F(1, 16) = 2.67$, $p < .10$, Expertise × Appropriateness of Set interaction ($F < 1$), Abstraction of Set × Appropriateness of Set interaction ($F < 1$), and Expertise × Abstraction of Set × Appropriateness of Set interaction, $F(1, 16) = 2.00$, $p > .10$, were not significant.

The results of the inappropriate set conditions are presented here followed by the results of the appropriate set conditions. The results of the inappropriate set conditions, where the level of abstraction of the flowchart and the question do not match, give us information relevant to some of the questions that were posed at the beginning of this article. In the inappropriate set conditions we can see what level of representation is most natural to novices and experts. The rationale is that in these conditions, the flowcharts inappropriately prepare the subjects to represent the code at one level of abstraction or another. If the level that actually is appropriate is their natural or preferred level, then they would still be able to perform well. That is, they would be able to quickly and accurately form a representation of the program that is appropriate for answering the question, and they would be able to process the information in it despite the misleading set. However, if the appropriate level is not natural to them, the effects of inappropriate set plus nonnaturalness should combine, and as a result, their performance should be impaired. So the pattern of results in the inappropriate set conditions would give us information about the natural level of representation for each group. A condition in which a group's performance is good is a condition in which the required representation is natural, and a condition in which a group's performance is poor is one in which the required representation is not natural. The means and standard errors for each condition are presented in Table 1. In order to focus on the conceptually interesting questions, the means are broken down by abstraction of question.

Inappropriate set conditions. Figure 5 is a visual representation of the means for the inappropriate set conditions in Table 1. Looking at the right-hand side of the figure, the Experts do better than the Novices when answering an abstract question. However, on the left-hand side, the Novices are better than the Experts when a concrete question has to be answered. The Expertise × Abstraction interaction for

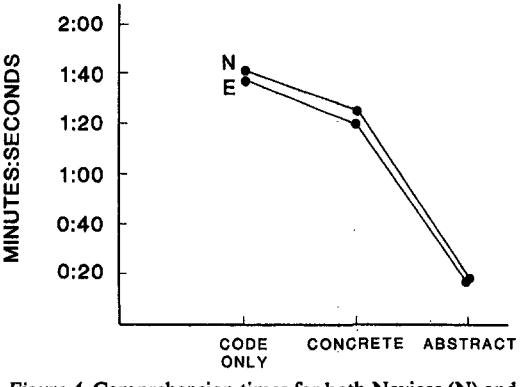

Figure 4. Comprehension times for both Novices (N) and Experts (E).

questions answered in the inappropriate set conditions was significant, $F(1, 8) = 6.00$, $p < .05$. The results of a Newman-Keuls test showed that the difference between the Novice and Expert groups was significant at the .05 level for both abstract and concrete questions. For error rates and response times, $r = .89$, $p < .05$. Recall that the current hypothesis is that novices form concrete representations whereas experts form abstract representations during program comprehension. Using the definitions previously given for abstract and concrete, this hypothesis then predicts that following a comprehension task, experts but not novices would have difficulty in answering concrete questions about how a program functions because the representation that they have formed no longer includes the details of how it functions. It also predicts that novices but not experts would have difficulty in answering abstract questions about what a program does because their representation does not include this type of information. The pattern of results found here supports the hypothesis. Although the results are striking, they are not counterintuitive when one is thinking about the underlying representations that are plausible for each group. The finding that nov-

Table 1
*Percentage Incorrect (*M *and* SE*) for Responses in Experiments 1 and 2*

	Condition					
	Inappropriate set		Appropriate set		Code only	
Question	*M*	*SE*	*M*	*SE*	*M*	*SE*
Experiment 1						
Concrete						
Novices	8	2	13	2	8	2
Experts	17	2	8	2	17	4
Abstract						
Novices	17	4	13	3	17	2
Experts	4	2	4	2	8	3
Experiment 2: Immediate						
Concrete						
Novices	50	8	38	6	—	—
Experts	69	5	38	17	—	—
Abstract						
Novices	38	6	44	10	—	—
Experts	13	6	38	6	—	—
Experiment 2: Delayed						
Concrete						
Novices	38	6	38	6	—	—
Experts	69	5	56	5	—	—
Abstract						
Novices	56	5	31	10	—	—
Experts	13	6	31	5	—	—
Experiment 2: Immediate and delayed						
Concrete						
Novices	—	—	—	—	25	3
Experts	—	—	—	—	50	4
Abstract						
Novices	—	—	—	—	50	4
Experts	—	—	—	—	25	3

Note. — = data not applicable.

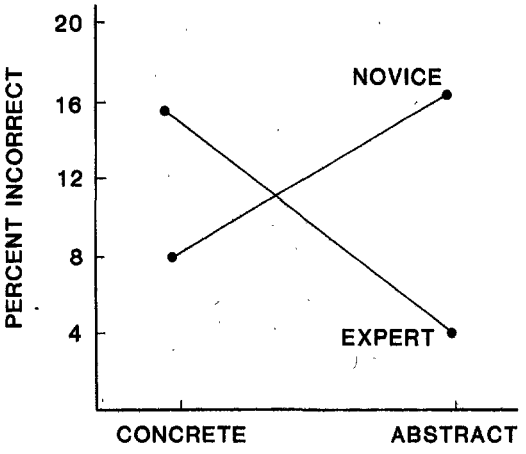

Figure 5. Error rates in the inappropriate set conditions.

ices produce concrete representations whereas experts produce abstract, conceptual ones is plausible and has been obtained by others as well (Adelson, 1981c; Chi et al., 1981; Simon, 1979).

In the code only condition, although the interaction did not reach significance, the pattern of Experts surpassing Novices on abstract questions and Novices surpassing Experts on concrete questions was again obtained for error rates ($F < 1$). (See Table 1.) For error rates and response times in the code only condition, $r = .58$, $p < .05$. The effects of expertise and abstraction of question were not significant ($Fs < 1$).

The results of the inappropriate set conditions suggest that there is a different preferred level of representation for each group and that the representation imposes some limitations on performance. At this point it is interesting to ask how strongly these performance limitations are fixed. Although it seems difficult for both groups to switch from their preferred levels of representation when they have been encouraged to use them, it is still possible that both groups can form representations at their nonpreferred levels that they can then use effectively if they are given the proper aid. This is what was done in the appropriate set conditions.

Appropriate set conditions. Figure 6 is a visual representation of the means in the appropriate set conditions (see Table 1). The Expertise × Abstraction interaction for questions answered in the appropriate set conditions was not significant ($F < 1$). A Newman-Keuls test showed that abstract and concrete questions were not differentially difficult at the .05 level both for the Novice and the Expert groups; however, the Novices found the abstract question more difficult than the Experts did. For error rates and response times, $r = .95$, $p < .05$.

The data suggest each group can be flexible in the kinds of representations they can form and use effectively. It is interesting that, as we saw in the inappropriate set conditions, the Novices do not tend to generate an abstract representation on their own; however, once they are aided in forming an abstract representation, they are able to use it, as we see here. The evidence supporting the first claim is that in the inappropriate set conditions, the Novices did significantly worse on abstract compared with concrete questions (at the .05 level using a Newman-Keuls test). The second claim is supported by the finding that the Novices do equally well on abstract and concrete questions in the appropriate set conditions.

This may parallel the "production deficiency stage" that is discussed in developmental literature (Flavell, 1977, p. 197). At this stage a child is able to use a strategy to improve his or her performance, but the strategy must be suggested by the experimenter; the child does not adopt it on his or her own. Before this stage, the child cannot adopt the strategy, and after this stage, his or her use of

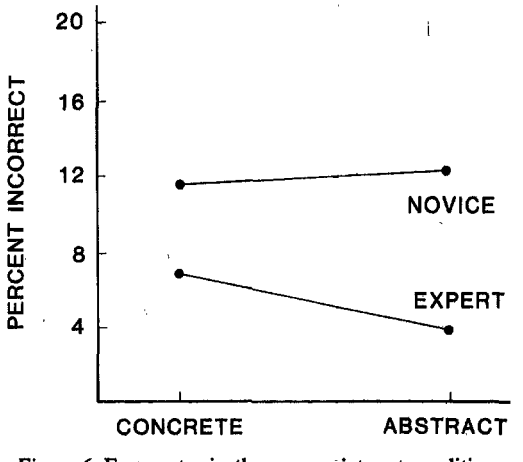

Figure 6. Error rates in the appropriate set conditions.

the strategy is spontaneous. The similarity here is that novices are able to adopt an appropriate set when it is suggested to them. The dissimilarity is that in this experiment, there is no consistently appropriate strategy that a more experienced subject can adopt spontaneously and can reliably profit from because the level of abstraction of the question varies randomly. Therefore, the experts, who spontaneously adopt the usually helpful strategy of abstractly encoding the program, suffer when the concrete questions come along. Why experts spontaneously form an abstract representation and why this is usually advantageous in comprehension tasks is discussed subsequently. However, it is interesting to note here that the experts do not lose the ability to form a concrete representation, that is, in the appropriate set conditions they do not perform differently on abstract and concrete questions. This point also is expanded on later.

Experiment 2

The results of the appropriate set conditions suggest that when subjects are given an appropriate set, their performance on tasks at their nonpreferred level of problem solving can improve to the point where it may equal their performance on tasks at their preferred level. This is supported by the difference between abstract and concrete questions for both Novices and Experts (significant by a Newman-Keuls test at the .05 level) in the inappropriate set conditions and the lack of significant differences in the appropriate set conditions. Even though we can see that the subjects do have some flexibility, how far does it go? Do the new representations formed by each group last or does each group remember only what they are used to remembering? Experiment 2 looked into this question. It also cleared up some points of uncertainty that resulted from the methodology of Experiment 1 while replicating its results.

Method

Subjects

A new group of 24 Novice and 24 Expert subjects served as paid volunteers. As before, the Novices were Harvard undergraduates who had completed an introductory course in computer programming, and the Experts were teaching fellows for that course.

Stimuli

Once again, eight PPL programs were used. This time the programs were slightly longer and more modular. Once again an abstract or a concrete set was established for the subjects, but it was established differently from Experiment 1. Subjects simultaneously saw the program along with an abstract or concrete task. In the abstract task, a main program was presented with its subroutines below it; however, one line had been deleted from the main program, and it was presented to the subject at the bottom of the program. The subject's task was to place this line properly in the main routine. (The subroutines were presented in an order that would not provide clues as to where the missing line belonged.) Because the abstract task had the subject focus on the flow of control of the program, it was a task that encouraged the subject to represent only what the program did. The abstract tasks of Experiments 1 and 2 were therefore similar in the way in which they fostered abstract representations.

The concrete task had the subject focus on the details of how the program functioned. In this task, the subject was told that each program had one or two bugs that had to be located. (For readers who are unfamiliar with the terminology, errors in a computer program are called *bugs;* the process of finding them is called *debugging.*) Because the concrete task had subjects concentrate on the details of how the program functioned, it was thought to foster concrete representations in a way that was similar to the concrete task of Experiment 1. Two of the eight programs had two bugs; the rest had only one bug. The subjects however, didn't know which or how many of the programs had more than one bug. In addition, the bugs were placed randomly so that they did not occur predictably early or predictably late in the programs. This encouraged the subjects to search through each entire program. The bugs were conceptual rather than syntactic, but they were contained within a single line. An example of a typical bug would be an incorrect test for the termination of a loop (e.g., if the program said "while i > 10" when it clearly should have said "while i < 10"). Bugs were not of the missing punctuation variety nor were they incorrect but executable algorithms.

Design

The level of abstraction of the task presented with each program was crossed with the level of abstraction of the question to form four conditions for 16 Novice and 16 Expert subjects: two appropriate set conditions and two inappropriate set conditions.

Novice and Expert subjects were divided into two groups, with one group of Novices and one group of Experts receiving the question immediately after having seen the program with its task (the immediate conditions) and the other group of Novices and Experts being given a 1.5-min distractor interval between seeing the program with its task and answering the question about it (the delayed conditions).

Error rates were recorded for the tasks and the questions. Two times were recorded: task time (the time it took the subject to say he or she was ready to begin to write an answer to the task) and question time (the time interval between seeing the question and being able to begin to write down the answer). The correlation between question

time and the number of errors on questions was used to investigate the presence of speed–accuracy trade-offs.

Each subject received tasks at only one level of abstraction; however, all subjects were asked questions that were both appropriate and inappropriate to the level of their task (four of each). Each subject saw only one question for each of the eight programs. Programs, questions, and question types were counterbalanced across subjects and across conditions. The ordering of appropriate and inappropriate questions was random. Subjects within a given level of expertise were assigned randomly to a task level of abstraction and a delay group.

Procedure

Experiment 2 used basically the same procedure as Experiment 1 but with a few important methodological changes. Each of the eight stimulus programs was presented one at a time to each subject. Rather than presenting abstract and concrete flowcharts before each program, an abstract or a concrete task was presented simultaneously with each program. The purpose of the manipulation was to establish an abstract or a concrete mental set for the subject.

After completion of the set-establishing task the program was removed, and subjects were then shown a question about the program that they had to answer without access to the program. Once again the question was either an abstract one or a concrete one. Half of the subjects had a 1.5-min distractor interval (solving a Rubik's cube) in between completing the task and seeing the question, whereas the other half was shown the question immediately. A stopwatch was used to record times, and subjects verbally indicated their readiness to respond to both the task and the question.

To summarize, there were three main differences between Experiments 1 and 2:

1. In Experiment 2, half of the subjects were given a 1.5-min distractor task between seeing the program and answering the question about it. This allowed us to see the limits of the effectiveness of the nonusual representations that were established in the appropriate set conditions.

2. In Experiment 2, the set was established by the task rather than by a flowchart, which cleared up one of the methodological problems of the last experiment, because subjects now had only one source of information (the program) to use in forming their representation. The possibility of different subjects having formed their representations from different materials (i.e., some using the flowchart and some using the program) no longer existed. As a result, any differences that were found in the inappropriate set conditions between groups with different levels of expertise arose from the differences in the groups' natural way of representing the material.

3. In Experiment 2 we also ascertained whether the performance differences of Experiment 1 were due to differences in the ability of the two groups to search through the code (because the code was no longer present when these subjects were answering the question). This cleared up the second methodological problem of Experiment 1 and tested the replicability of its results.

Code only conditions were again included. In the immediate code only conditions, 4 Novices and 4 Experts studied the program until they indicated that they understood it; they then immediately saw the question appropriate to their condition. In the delayed code only conditions, this procedure was repeated with 4 Novices and 4 Experts; however, a 1.5-min distractor task intervened between seeing the program and seeing the question.

Table 2
Response Times (RT in min:s) and Error Rates (% Incorrect) for Experiment 2

Task group	RT	Error rates
Novices		
Concrete	11:06	84.44
Abstract	4:42	44.44
Code only	5:40	—
Experts		
Concrete	8:54	21.88
Abstract	3:45	15.63
Code only	4:00	—

Note. — = data not applicable.

Results and Discussion

Task Results

Task response times are presented in Table 2. The effects of expertise, $F(1, 24) = 6.28$, $p < .05$, and task, $F(1, 24) = 127.29$, $p < .05$, were both significant. The Abstraction × Expertise interaction was not significant, $F(2, 24) = 2.10$, $p > .10$.

Error rates also are presented in Table 2. The effects of expertise, $F(1, 24) = 29.67$, $p < .01$, and task, $F(1, 24) = 10.96$, $p < .05$, were significant, as was the Task × Expertise interaction, $F(1, 24) = 7.13$, $p < .05$. The debugging task was more open-ended than the flow of control task because many types of bugs were possible even within the restricted range of the bugs used, and this may have contributed to its difficulty.

Responses to Questions

The effects of expertise ($F < 1$); delay ($F < 1$); abstraction of task, $F(1, 16) = 3.19$, $p > .09$; and appropriateness of set, $F(1, 16) = 1.19$, $p > .10$, were not significant. The only significant second-order interactions were Expertise × Abstraction of Task, $F(1, 16) = 5.45$, $p < .05$, and Abstraction of Task × Appropriateness of Set, $F(1, 16) = 25.19$, $p < .05$. The only significant higher order interaction was Expertise × Abstraction of Task × Appropriateness of Set, $F(1, 16) = 17.19$, $p < .05$.

Inappropriate set conditions: delayed and immediate groups. The error rates for subjects

who had no delay are presented in Table 1. When a concrete question has to be answered, the Novices make fewer errors than the Experts; when an Abstract question has to be answered, the Experts make fewer errors than the Novices. The Expertise × Abstraction of Question interaction is significant here, $F(1, 8) = 32.81$, $p < .001$, and the results of a Newman-Keuls test showed the difference between the Novice and the Expert groups was significant at the .05 level for both abstract and concrete questions. The error rates for subjects who had both an inappropriate set and a delay are also presented in Table 1. When a concrete question has to be answered, the Novices again make fewer errors than the Experts; when an Abstract question has to be answered, the Experts make fewer errors than the Novices. The Expertise × Abstraction of Question interaction is significant, $F(1, 8) = 32.81$, $p < .001$, and the results of a Newman-Keuls test showed the difference between the Novice and the Expert groups was significant at the .05 level for both abstract and concrete questions. For error rates and response times in the inappropriate set conditions, $r = .68$, $p > .05$.

In the code only conditions, the pattern of Experts surpassing Novices on abstract questions and that of Novices surpassing Experts on concrete questions was again obtained. The Expertise × Abstraction of Question interaction was significant, $F(1, 8) = 11.57$, $p < .01$. (See Table 1.) No other main effect or interaction reached significance at the .05 level. For error rates and response times, $r = .32$, $p < .05$.

Appropriate set. The main effects and interactions of expertise, delay, and abstraction were not significant [all Fs < 1 except for the Delay × Abstraction of Question interaction, $F(1, 8) = 1.42$, $p > .05$]. For error rates and response times, $r = .78$, $p < .05$.

Appropriate set: immediate group. Looking at the error rates presented in Table 1 for the immediate condition with appropriate set, when the task and the question match and there is no delay between them, Novices and Experts showed no significant difference on either abstract or concrete questions. Using a Newman-Keuls test, no significant differences were found at the .05 level. This again suggests that both groups can be flexible in the kinds of representations they can form and use.

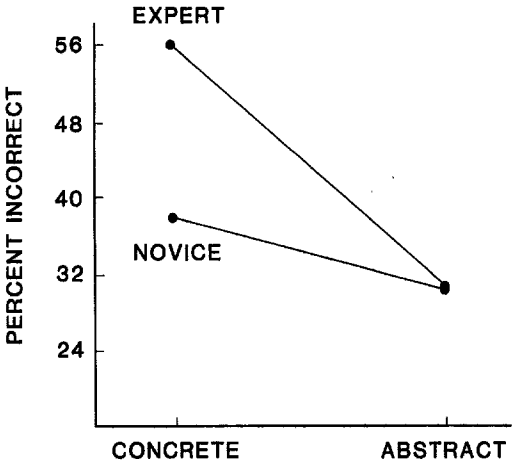

Figure 7. Error rates in the appropriate set conditions for the delayed group.

Appropriate set: delayed group. Although the results for the immediate group suggested that both groups can perform well when the proper set for a task is established beforehand, in the delayed conditions, where subjects did not see the program and the question simultaneously, the effect of the set cannot entirely overcome the subject's predisposition.

The mean error rates presented in Table 1 for the delayed appropriate set conditions suggest that after a delay, the pattern of results begins to return to what it was in the inappropriate set conditions; the Novices make fewer errors than the Experts on the concrete questions (a Newman-Keuls test was significant at the .05 level). The results do not perfectly match those from the inappropriate set conditions; when an abstract question has to be answered, the Experts and the Novices do equally well. These data suggest that after a brief delay, each group tends to remember only aspects of the program that they are accustomed to remembering. This would be the case if in the appropriate set conditions, subjects initially formed both the set-induced representation and their naturally preferred type of representation (in the cases where these two differ) and then had an easier time in transferring into or accessing from long-term memory the type of representation to which they were accustomed. This process, of initially forming both types of representations and later storing or retrieving only the natural one, can account for the results in the immediate and the delayed appropriate set conditions. In the

immediate conditions, the subjects have their set-established representation along with the one that they spontaneously form, and so they do equally well on abstract and concrete questions. In the delayed conditions, they more reliably have their preferred representations, and so they do better on those questions that are about them (see Figure 7).

Summary and Conclusions

It is striking to find novices surpassing experts; however, that fact is important only in so far as it points out what the problem representations of the novice and the expert are during program comprehension. The data suggest the representation of the expert is more abstract and contains more general information about what the program does, whereas the representation of the novice is more concrete and contains information about how the program functions. The methodology used here shows this by choosing a set of two tasks, in which one is well-suited for the novices but not the experts, and the other is well-suited for the experts but not the novices. This allows us to see both the utility and the limitation of each type of representation. The data also suggest that mental sets can be established, enabling each group to represent and use information in a nonusual way; however, the effect of the set is limited in that the pattern of each group's performance tends, over time, to revert to the pattern that we find in the absence of any experimentally induced set.

The results of these experiments do not suggest that expert programmers lose the ability to attend to the details of a program. (In the appropriate set conditions in which there is no delay, we see that the Experts are quite good at attending to detail.) Rather, they suggest that experts have learned that, during comprehension of this type of program, paying attention to the abstract elements of the program is more important than paying attention to the low-level details. Why this is the case is discussed shortly. This point can hold even for non-Algol and/or very high-level languages in which the distinction between what is abstract and concrete may not map onto the distinction between what the program does and how the program functions.

The literature reviewed in the beginning of this article suggested that experts formed abstract representations. However, it is interesting to look more closely at this kind of finding in light of the specific nature of the abstract representations found here. Jeffries, Turner, Polson, and Atwood (1981) and Atwood and Jeffries (1980) collected protocols from computer scientists as the scientists were designing an indexing program that would create a list of the occurrences of a given set of key terms in a given text. Consider one representative protocol in which the subject stated in her initial representation that the solution to the problem would consist of three main procedures. The three procedures were: Read in the set of key terms, compare the key terms to the text, and store the resulting index. (Although the subject did not mention it, a procedure to read in the text would have to be included as well.) This protocol seems representative of the ones collected, and the way that the problem is stated does suggest that the experts represented the problem statements in terms of what the program would do. Another instance of this kind of representation is found in Atwood, Turner, Ramsey, and Hooper (1979). They showed expert programmers' specifications for to-be-designed programs. The programmers were then asked to summarize these specifications. An analysis of the structure of the summaries produced by the subjects suggested that their organization reflected the main processes of the to-be-designed programs (Kintsch, 1974).

Instances of experts representing problems in terms of what is to be done but not how to do it is not limited to the domain of computer science. Brown and Burton (1978) found that elementary school math teachers, in acquiring their arithmetic skills, had acquired "macro" or unanalyzed procedures. That is, although they were expert in performing arithmetic operations, they had difficulty in verbalizing what these operations were. This appears to be another case in which a group of experts have come to know what they do without having access to detailed representations of how they do it. Anderson, Greeno, Kline, and Neves (1981) and Neves and Anderson (1981) suggested a model for the task of generating formal proofs. It seems that their model would predict the current finding. In their model, information that is well known to the subject can be represented as a procedure that can be used to match, and thereby recognize, incoming information very rapidly. However, knowledge contained in a procedure

cannot be inspected directly; what the knowledge is can only be inferred by noting what the procedure does. Having developed these procedures, the information comes to be represented in a way that hides the details of the processing to be done. Although Anderson is developing a mechanism for recognizing information, whereas I am focusing on the format of the information being recognized, we both suggest that in some situations experts will represent information in a way that hides the details of the processes.

Given that this type of representation does occur across a number of problem-solving domains, we can ask, What is its utility? In the above-mentioned study by Jeffries et al. (1981), in which protocols were collected of experts designing a book indexing program, one expert stated early on in her protocol that one of the three main procedures in the problem solution would be a procedure in which the key terms would be compared with the text. She later went back and compared alternative methods for doing this without mentioning the other aspects of the problem that she had initially identified as separate processes. This suggests that representing problems in terms of several elements that describe what is to be done is useful, because it later allows the expert to elaborate or to change an element in isolation without exceeding the limits of working memory.

To conclude, previous research had shown that experts formed abstract representations; however, the issue of what information was contained in these representations was not addressed directly. As a result, it was not known why abstract representations were of value to the expert while solving a problem. The data presented here address this question. They suggest that experts represent the problems that they are solving as elements that describe the operations to be performed. The utility of these elements is that they are easy to work with and easy to change and thereby allow the expert to find an optimal solution to a problem.

References

Adelson, B. (1981a, August). *Cognitive spaces for computational categories.* Paper presented at the meeting of the Cognitive Science Society, Berkeley, CA.

Adelson, B. (1981b). Knowledge structures of computer programmers. *Proceedings of the Fourth Annual Meeting of the Cognitive Science Society, 4,* 243–248.

Adelson, B. (1981c). Problem solving and the development of abstract categories in programming languages. *Memory & Cognition, 9,* 422–433.

Anderson, J. R., Greeno, J. G., Kline, P. J., & Neves, D. M. (1981). Acquisition of problem-solving skill. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 141–191). Hillsdale, NJ: Erlbaum.

Atwood, M., & Jeffries, R. (1980). *Studies in plan construction* (Tech. Rep. No. SAI-80-028-DEN). Englewood, CO: Science Applications.

Atwood, M., Turner, A., Ramsey, R., & Hooper, J. (1979). *An exploratory study of the cognitive structures underlying the comprehension of software design problems* (Tech. Rep. No. SAI-79-100-DEN). Englewood, CO: Science Applications.

Bhaskar, R., & Simon, H. A. (1977). Problem solving in semantically rich domains. *Cognitive Science, 1,* 193–215.

Brown, J. S., & Burton, R. R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science, 2,* 155–192.

Chase, W. C., & Simon, H. A. (1973). Perception in chess. *Cognitive Psychology, 4,* 55–81.

Chi, M., Glaser, R., & Rees, E. (1981). Expertise in problem solving. In *Advances in the psychology of human intelligence* (Vol. 1). Hillsdale, NJ: Erlbaum.

de Groot, A. D. (1965). *Thought and choice in chess.* Paris: Mouton.

Flavell, J. H. (1977). *Cognitive development.* Englewood Cliffs, NJ: Prentice-Hall.

Jeffries, R., Turner, A., Polson, P., & Atwood, M. (1981). The processes involved in designing software. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition:* Hillsdale, NJ: Erlbaum.

Kintsch, W. (1974). *The representation of meaning in memory.* Hillsdale, NJ: Erlbaum.

Lewis, C. (1981). Skill in algebra. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition:* Hillsdale, NJ: Erlbaum.

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

Neves, D. M., & Anderson, J. R. (1981). Knowledge compilation: Mechanisms for the automatization of cognitive skills. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 57–84). Hillsdale, NJ: Erlbaum.

Reitman, J. S. (1976). Skilled perception in go: Deducing memory structures from inter-response times. *Cognitive Psychology, 8,* 336–356.

Ryle, G. (1949). *The concept of mind.* London: Hutchinson.

Schneiderman, B. (1977). *Measuring computer program quality and comprehension* (Tech. Rep. No. 16). College Park: University of Maryland, Department of Information Systems Management.

Simon, H. A. (1979). Information processing models of cognition. *Annual Review of Psychology, 30,* 383–396.

Winograd, T. (1974). *Natural language understanding,* New York: Academic Press.

Received July 20, 1982
Revision received October 12, 1983 ■