1985

Expertise in Debugging Computer Programs: Situation-Based versus Model-Based Problem Solving

Iris Vessey
University of Queensland

Expertise in Debugging Computer Programs: Situation-Based versus Model-Based Problem Solving

Iris Vessey
University of Queensland

Acknowledgment: The author is indebted to Ron Weber for his assistance throughout this research.

ABSTRACT

This paper reports the results of an exploratory study that investigated expert and novice debugging processes with the aim of assessing the relevance of situation-dependent problem solving to debugging expertise. The method used was verbal protocol analysis. Data was collected from sixteen subjects employed by the same organization. The study first controlled for the variability in individual problem solving by incorporating certain aspects of programmers' debugging processes into the debugging model. The criterion of expertise was the subjects' ability to effectively chunk the program they were required to debug. This method proved effective in explaining much of the variability in debugging performance and provided the basis for the expert-novice classification used in subsequent analysis of the protocol data. Further analysis focused on situational factors in debugging. It took two forms: (1) a static or content analysis of subjects' problem solving behavior that aggregated data across a protocol; and (2) a dynamic or process analysis of subjects' debugging processes that examined data as closely as possible to its natural state. The results support the notion that experts respond to the data in the task while novices are constrained by preconceived ideas or early hypotheses about the source of error.

KEYWORDS: Debugging; programmer skills; situation-dependency; debugging processes; protocol analysis

Introduction

This study examined the debugging processes of expert and novice programmers with the aim of identifying those characteristics that lead to expertise. The characteristic of expertise addressed primarily in this research was the extent to which problem solvers rely on formal models in solving problems as opposed to relying on data derived from the task. This characteristic of expertise is termed situation-dependency. Using the process tracing technique of recoding verbal protocol (Newell and Simon, 1972; Ericsson and Simon, 1980, 1984), the research examined both the content of expert-novice problem solving and the processes experts and novices use.

Previous programming studies have reported high variability in programmer performance that frequently masked the effects of the manipulated variables (Mayer and Stalnaker, 1968 Brooks, 1980; Sheil, 1981; Pennington, 1982). Hence, this study sought to identify problem solving characteristics that accounted for much of that variation in performance. Once identified, these aspects of programmers' problem solving performance were controlled by means of the expert-novice programmer classification. Removing some of the variability in programmers' processes then permitted the effect of situation-dependency to be tested more effectively.

The paper proceeds as follows. The next section describes the conceptual approach followed in this study. A detailed task analysis based on literature in computer science and cognitve psychology and on information derived from pilot study protocols formed the basis for analysis for subject protocols. The methodology section describes the task materials, subjects, and the performance measures used in this study; it also addresses the

question of how we determine which programmers are experts and which are novices, a distinction that plays a central role in this analysis of debugging processes; the section concludes with a brief description of the protocol analysis methodology used and associated reliability measurement undertaken. The following section presents the results of both the content and process analyses. Finally, the paper disucsses the results and examines the implications of the results for the concept of situation-dependency.

Conceptual Framework

According to Dreyfus (1982), novices use a model-based approach to solving a problem, while experts draw on their experience of familiar situations. The use of an internal model implies the application of formal rules. Hence, this approach is necessarily independent of the particular situation, i.e., it is context-free (model-driven problem solving). With experience, this understanding is transformed into a superior type of understanding, one that is situation-dependent or data-driven. Experts are those who conceive of the problem in context. They do not work through a set of formal rules derived from a prespecified model. Rather, the solution emerges from their characterization of the problem in terms of their previous experience with similar kinds of problems.

Situation-dependent of data-driven problem solving is manifested in high-level problem solving (Heller and Greeno, 1978). Problem solvers who engage in situation-dependent problem solving tend to spend more time in understanding and formulating a problem than do novices (Reitman, 1965). Novices, instead, need to convince themselves that they are making progress. They therefore close the constraints on the problem as soon as possible and attempt to force a solution even to the extent of not acknowledging disconfirming evidence (Rouwman, 1978).

The study examines both the types of behavior (content of debugging) and the processes programmers engage in while debugging. Both the content and process analyses address the importance of situation-dependency in problem solving. The content analysis does so, formally, by developing two content descriptions, one for a model-based approach, and the other for a situation-dependent approach. The first content description, referred to as debugging functions, was based on a procedural (flowchart) model of debugging that was converted to a static function model for analysis (Figure 1). The second content description, referred to as debugging activities, derives from literature in computer science on debugging and in cognitive psychology on problem solving, supplemented with activities present in pilot study protocols. Figure 2 shows the structure of the debugging activities. Debugging activities are behaviors known to be present in debugging and are not related to a model of the debugging process. They are, therefore, situation-dependent. Both the debugging functions and the debugging activities were structured hierarchically to permit differing degrees of sensitivity in the analysis. Partitioning also has methodological implications in that certain function or activity categories might contain too few or too many responses; the appropriate level can thus be chosen for further investigation. Propositions were derived for the effect of expertise on each of the debugging functions and debugging activities. The propositions reflected the expectation that differences in expert-novice behavior would become more apparent with more complex tasks.

The process analysis used uncoded data so the data was not constrained to meet the requirements of a model. In this way, the researcher is able to take into account characteristics of the data as they arise. Hence, from the viewpoint of testing whether novices are situation-independent problem solvers while experts are situation-dependent problem solvers, the process analysis permits situation-dependent characteristics to be examined better than does the content analysis. The process analysis sought to determine differences in the ways expert and novice programmers accomplished the problem solving task, approached the problem solving process, and pinpointed the program error. A further outcome of the process analysis was a characterization of the debugging strategies used by expert and novice programmers.

Research Methodology

The research methodology used in this study was verbal protocol analysis. Subjects were given an incorrect program listing, a copy of some input data, the correct output, and the corresponding incorrect output, and were asked to debug the program, speaking aloud as they did so. They debugged by means of desk checking, i.e., they did not have automated aids available to them.

DEBUGGING TASKS

The debugging environment was the maintenance section of a commercial data processing department. Hence subjects debugged programs they had not previously seen. The program used was a fully structured, straightforward COBOL sales reporting program with control breaks on branch number, salesperson number, and customer number (see Appendix A.5 of Vessey, 1984). A simple application domain was used so that differences in application domain knowledge hopefully would not affect the results, thus permitting the investigation of debugging processes alone.

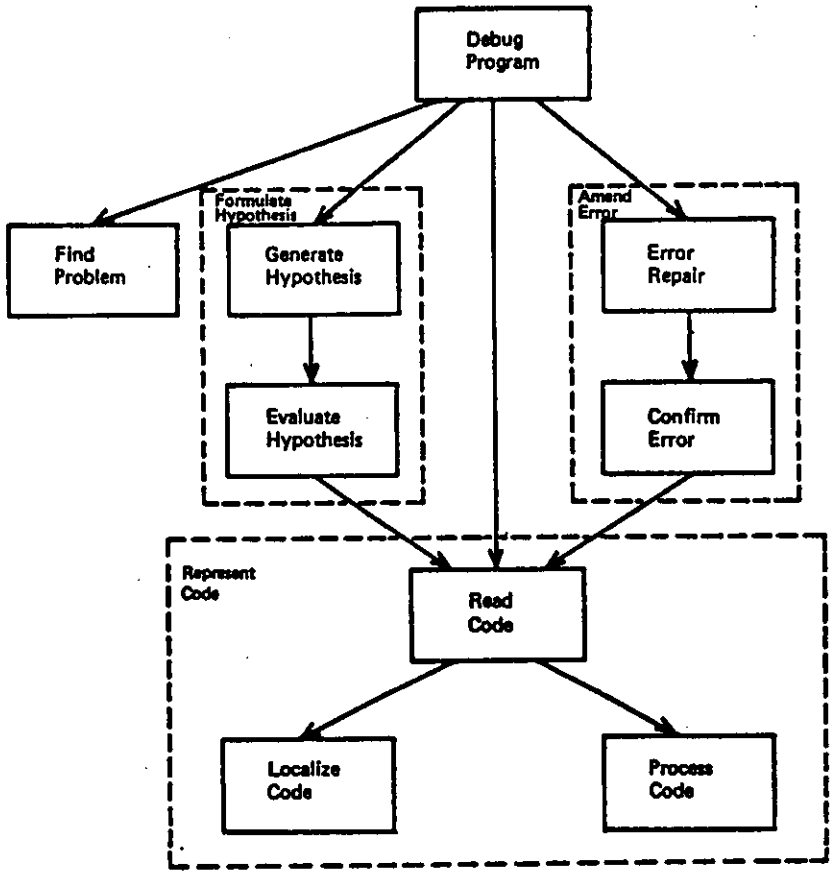

Figure 1

Model of Debugging Functions

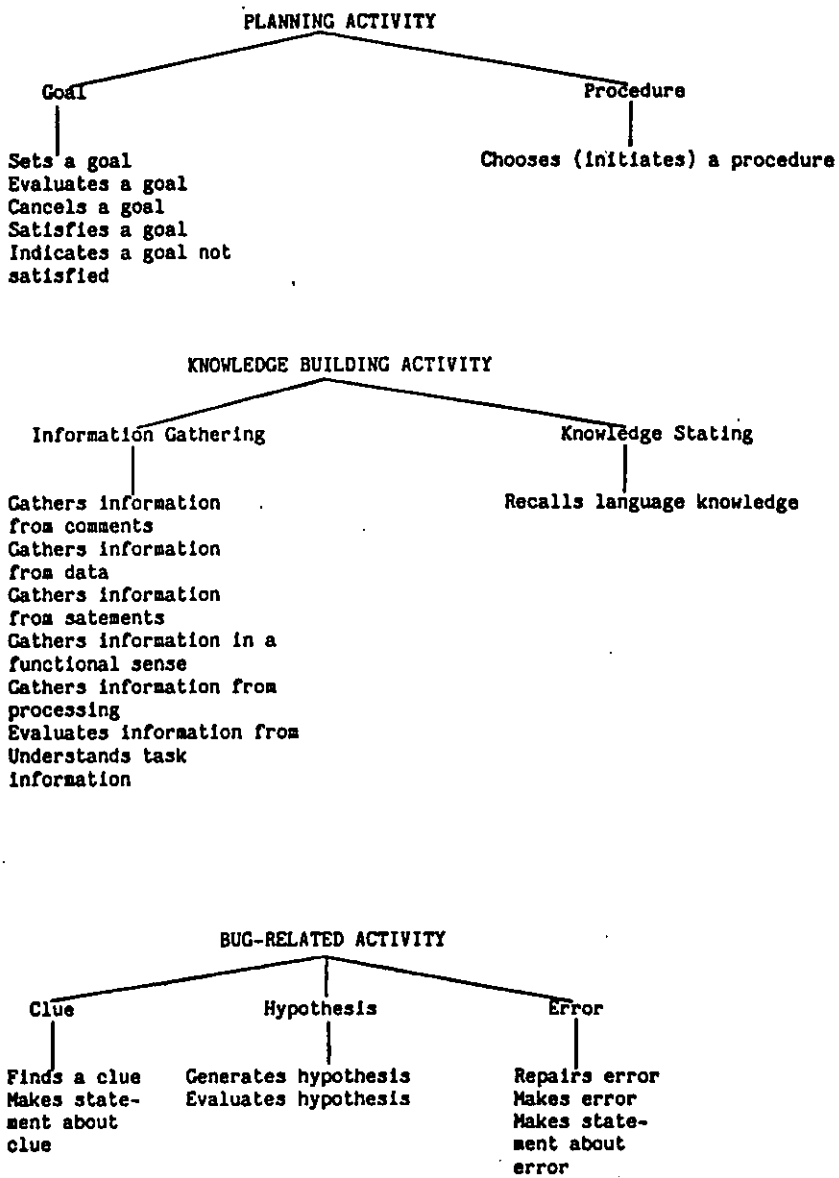

Figure 2

Structure of Debugging Activities

PROGRAM BUGS

The error introduced into the program was a logic error, a type commonly found in practice (Youngs, 1974; Gould and Drongowski, 1974; Gould, 1975; Sheppard *et al.*, 1979). No syntactic errors were present. As a basis for determining whether the task was sufficiently difficult to differentiate between experts and novices, the "same" bug was introduced at different locations in the program. Research by Atwood and Ramsey (1978) suggests that debugging complexity increases when the same bug is both lower in the sentence structure and lower in the program structure. Accordingly, this research used two program versions, one with the bug higher in both the sentence structure and the program structure, and the other with the bug lower in both structures. The correct program logic is as follows:

```
0295  IF BRANCH-CHANGE EQUALS "YES"
0296    MOVE BRANCH-NO-INPUT TO
          BRANCH-NO-REPORT
0297    MOVE SALESMAN-NO-INPUT TO
          SALESMAN-NO-REPORT
0298    MOVE CUSTOMER-NO-INPUT TO
          CUSTOMER-NO-REPORT
0299    MOVE 'NO' TO BRANCH-CHANGE
0300  ELSE
0301    IF SALESMAN-CHANGE EQUALS
          'YES'
0302      MOVE SALESMAN-NO-INPUT TO
            SALESMAN-NO-REPORT
0303      MOVE CUSTOMER-NO-INPUT TO
            CUSTOMER-NO-REPORT
0304      MOVE 'NO' TO
            SALESMAN-CHANGE
0305  ELSE
0306    IF CUSTOMER-CHANGE EQUALS
          'YES'
0307      MOVE CUSTOMER-NO-INPUT TO
            CUSTOMER-NO-REPORT
0308      MOVE 'NO' TO CUSTOMER-
            CHANGE.
```

The high-level bug was introduced into the program by removing line 299, which resets the branch-change flag; and the low-level bug by removing line 308, which resets the customer-change flag, and placing the period at the end of line 307.

SUBJECTS

The sixteen programmers who participated in this research were practicing programmers from the State Government Computer Center (S.G.C.C.), Brisbane, Australia. This sample size is large for a methodology that is both tedious and time-consuming. With one exception, all the programmers had spent their entire programming careers at the S.G.C.C. One person had spent two years at another government institution and at the time of the study had been employed by the Center for fifteen months. Thus, the subjects had homogeneous backgrounds.

PERFORMANCE MEASURES

Two types of performance criteria were established: effectiveness and efficiency criteria. They are as follows:

Effectiveness

Debug time, adjusted for verbalization rate
Number of mistakes made

Efficiency

Number of changes in debugging functions
Number of reversals to the Debug Program function
Number of changes in program location

The effectiveness criteria were used to identify which of the expert-novice programmer classifications tested best captured programmer expertise. The efficiency criteria "described" the ease with which programmers approached the task and thus provided the basis for a classification of programmers. The first two efficiency factors derived directly from the function analysis. Program locations were those units of task material regarded as chunks for the purposes of this research; they were the DATA DIVISION, modules of the PROCEDURE DIVISION, the input, and the correct and incorrect outputs. It was expected that novices would exhibit more of all (five) types of performance behavior than novices.

ASSESSING DEBUGGING EXPERTISE

In an attempt to better understand the nature of programmer expertise, two measures of expertise were investigated: an *ex ante* method—the expert opinion of managers at the S.G.C.C. (Reilly *et al.*, 1975)—and an *ex post* method derived from the protocol data collected in this study. The latter method relied on programmers' chunking skills to differentiate experts from novices. It used the three efficiency measures as indicators of chunking ability. These variables reflect the ability of programmers to chunk the program by assessing the extent to which problem solving is a smooth-flowing procedure as opposed to an erratic or disorganized one. Greater chunking ability, a property of experts, is manifested in fewer changes in the three efficiency measures. The analysis for program location changes controlled for bug level since bug level

significantly influenced the extent to which programmers examined different parts of the listings. Table 1 shows those subjects classified as expert and novice programmers by each of the programmer classifications. Only 10 of the 16 subjects were assigned to the same class by both methods. Hence, the correspondence between the classifications was low. The two classifications were first assessed on the basis of the debugging effectiveness measures, time and accuracy; next, the classification that produced the best time and accuracy figures was used to further investigate problem solving behavior.

VERBAL PROTOCOL CODING

Subjects in the debugging study were directed simply to speak aloud as they debugged one of the two program versions and the verbal protocol was tape recorded. This data was then converted to a form suitable for the content analysis by coding it according to a coding scheme based on the debugging functions and debugging activities. Tables 2(a) and (b) show the coding scheme variables and the expected direction of the results of testing the empirical propositions. Table 3 shows similar information for the performance measures.

Two independent coders who were naive to the psychological constructs under investigation scored the verbal protocols. Several reliabilty measures were calculated (see Tables 4.7 (a)–(e): Vessey, 1984). The reliabilities for all levels of all parts of the coding scheme were assessed as satisfactory compared with other studies. Multidimensional scaling (Young and Lewyckyj, 1979) showed that both coders scored in an essentially similar manner so that either coder's scoring was suitable for further analysis.

Data Analysis

Table 1 shows basic subject and task information. Note that the subject who both accomplished the task in the shortest time and spoke at the fastest rate had only two weeks' experience as a practicing programmer and was rated as a novice by managers. The data was analyzed using univariate analysis of variance (ANOVA). In all cases there were two factors: the programmer skill level derived from the particular classification used and the level of the program bug. For analyses involving debug time, analysis of covariance (ANCOVA) was used, the covariate being verbalization rate. The analysis controlled for verbalization rate since speaking aloud slows mental processes (Ericsson and Simon, 1980). The Newman-Keuls test was used to investigate further the effect of any interactions revealed by the ANOVA and/or ANCOVA models (Winer, 1971).

PERFORMANCE ANALYSIS

The first step in analyzing the protocol data was to determine whether (a) a change in bug location resulted in tasks that challenged programmers to different extents, and (b) which programmer classification best differentiated programmer performance. These analyses were carried out to ensure that both the task and the expert-novice classification permitted capturing characteristics of expertise.

Evaluation of Bug Complexity

Bug level was a significant determinant of debug time for both the manager and the *ex post* classifications ($p = .042$ and .001, respectively). As expected, the low-level bug required more time to detect and correct than the high-level bug. In addition, the result for the *ex post* classification showed a significant interaction effect manifested particularly in the behavior of novices with low bugs; they took significantly more time to complete the task than experts with low bugs and novices with high bugs. Hence, these results demonstrate that the program with the low-level bug was harder to repair than that with the high-level bug and that subjects were challenged by the debugging problem chosen as the task in this study. Subjects should, therefore, have manifested expertise in this task environment.

Evaluation of the Two Programmer Classifications

The manager classification (together with bug level) accounted for 36.1 percent of the variation in debug time, while the *ex post* classification accounted for 73.7 percent of the variation. Further, the *ex post* method outperformed the manager classification in classifying all (5) programmers who made mistakes as novices, compared with 4 out of 5 for the manager classification. On these grounds, the *ex post* classification method, was deemed to be more suitable for further analysis.

CONTENT ANALYSIS

The dependent variable tested in the content analysis (using 2-factor ANOVA) was the proportion of a particular behavior appearing in the protocol. Proportions were used since the number of phrases in the responses varied among subjects. Table 4 shows the significant results of testing the function and activity variables for the effects of the *ex post* programmer classification. Although there are a number of significant results, few hypotheses are supported due to the requirement that the

Table 1

Basic Subject Information

Subject [1]	Experience Months	*Ex ante* Classi-fication	*Ex post* Classi-fication	Bug Level	Time Mins:Secs	Words	Rate Words/Secs
EH1	22.0	Expert	Expert	High	11:00	891	1.35
EH2	12.0	Novice	Expert	High	17:47	837	0.78
EH3	11.0	Expert	Expert	High	14:43	1209	1.37
EH4	2.0	Novice	Expert	High	15:40	1230	1.31
NH1	2.5	Novice	Novice	High	20:50	2170	1.73
NH2	10.0	Expert	Novice	High	19:33	1447	1.23
NH3	24.0	Expert	Novice	High	21:40	1458	1.12
NH4	05.	Novice	Novice	High	17:20	1107	1.06
EL1	24.0	Expert	Expert	Low	19:23	1259	1.08
EL2	24.0	Expert	Expert	Low	25:29	1910	1.25
EL3	.05	Novice	Expert	Low	8.40	1047	2.01
EL4	40.0	Expert	Expert	Low	12:40	956	1.26
NL1	33.0	Expert	Novice	Low	38:44	2583	1.11
NL2	10.0	Novice	Novice	Low	31.38	2139	1.13
NL3	36.0	Novice	Novice	Low	36:46	2854	1.29
NL4	0.5	Novice	Novice	Low	37:54	3568	1.57

[1]Subjects are henceforth identified by codes. The first character identifies the subject as either an expert or a novice according to the *ex post* classification. The second character identifies the program bug as either a high-level or a low-level bug. Subjects are further identified, within theses classes, with a numeric character.

Table 2(a)

Debugging Function Hypotheses

Level	Code	Function	Hypothesis	Direction[1]
A	DB	Debug Program	HF8	N > E for L
	PF	Find Problem	HF9	E > N for L
	HF	Formulate Hypothesis	HF20	E > N for L
	EA	Amend Error	HF11	N > E for L
	CR	Represent Code	HF12	N > E for L
B	HFG	Generate Hypothesis	HF1	E > N for L
	HFE	Evaluate Hypothesis	HF2	E > N for L
	EAR	Repair Error	HF3	N > E for L
	EAC	Confirm Error	HF4	N > E for L
	CRR	Read Code	HF5	N > E for L
	CRL	Localize Code	HF6	N > E for L
	CRP	Process Code	HF7	E > N for L

[1]For ease of interpretation, the expected direction is recorded throughout as "A > B".

E = Expert
N = Novice
L = Low-level (more complex) bug

Table 2(b)

Debugging Activity Hypotheses

Level	Code	Activity	Hypothesis	Direction
A	P	Planning	HAP8	E > N for L
	K	Knowledge building	HAK10	N > E for L
	B	Bug-related	HAB11	N > E for L
B	GO	Goal	HAP6	E > N for L
	PR	Procedure	HAP7	E > N for L
	IG	Information gathering	HAK8	N > E for L
	KS	Knowledge stating	HAK9	N > E for L
	CL	Clue	HAB8	N > E for L
	HY	Hypothesis	HAB9	N > E for L
	ER	Error	HAB10	N > E for L
C	GOX	Sets a goal	HAP1	E > N for L
	GOE	Evaluates a goal	HAP2	N > E for L
	GOC	Cancels a goal	HAP3	N > E for L
	GOS	Satisfies a goal	HAP4	N > E for L
	GON	Indicates goal not satisfied	NAP5	N > E for L
	IGC	Gathers information from comments	HAK1	E > N for L
	IGD	Gathers information from data	HAK2	E > N for L
	IGS	Gathers information from statements	HAK3	N > E for L
	IGF	Gathers information in a functional sense	HAK4	E > N for L
	IGP	Gathers information from processing	HAK5	E > N for L
	IGE	Evaluates information	HAK6	E > N for L
	IGK	Understands task information	HAK7	E > N for L
	CLF	Finds a clue	HAB1	E > N for L
	CLS	Makes a statement about a clue	HAB2	N > E for L
	HYG	Generates hypothesis	HAB3	E > N for L
	HYE	Evaluates hypothesis	HAB4	E > N for L
	ERR	Repairs error	HAB5	N > E for L
	ERM	Makes error	HAB6	N > E for L
	ERS	Makes statement about error	HAB7	N > E for L

effect be manifested to a greater extent through the more difficult, low-level bug. The results were as follows.

Debugging Functions

Novices spent more time in the Debug Program function (DB); experts spent more time in the Find Problem function (PF); and novices spent more time in the Error Repair function (EAR). All results were in the expected direction but since the effects were not manifested to a greater extent through the harder, low-level bug, the relevant hypotheses HF8, HF9, and HF3 were not supported.

Planning Activities

The low frequency of occurrence of the significant variables, goal evaluating activity (GOE) and unsatisfied goal activity (GON), calls into question the stability of the results.

Knowledge Building Activities

The *ex post* skill classification significantly affected two knowledge building activities. First, experts exhibited more information evaluating activity (IGE) than novices; the result was in the direction postulated in hypothesis

HAK6, but the level of activity did not increase for the low-level bug and the hypothesis was not supported. Second, there were two significant interaction effects for information gathering from program statements activity (IGS). (a) Novices displayed more of the activity than experts for high bugs. Hypothesis HAP3 predicted that novice activity would exceed that of experts for the low bug. Apparently, experts required much less information search to find the high bug than did novices, while the differences for the low bug were not as marked. (b) Experts displayed more of the activity for low bugs than for high bugs.

Bug-Related Activities

There were three significant main effects for bug-related activities, but no interaction effects, so none of the hypotheses was supported. First, experts displayed more clue finding activity (CLF) than novices. Second, experts also displayed more clue activity (CL) than novices. It was expected that experts would exhibit more clue finding activity than novices but that the frequency of clue stating activity (CLS) would outweigh clue finding activity and that novices would, therefore, exhibit more aggregate clue activity than experts. Third, as expected, novices displayed more error making activity (ERM) than experts.

PROCESS ANALYSIS

The basis for examining problem solving processes was the episode: a group of task assertions related to the same goal or objective (Newell and Simon, 1972, p. 84). A subject's protocol comprises a sequence of such episodes, each associated with the fulfillment of a specific goal. Hence, the representation of a subject's protocol in episode form captures the goal-oriented behavior of the subject and the sequence in which it occurs. It can be used, therefore, as the backbone for the representation of the problem solving process.

Episodes can be aggregated into larger groups of related behaviors, known here as phases. All subjects' protocols could be characterized in terms of a limited number of such phases. Those phases found to be present in debugging protocols were problem determination, gaining familiarity with program function and structure, exploring program execution and/or program control, and repairing (and confirming) the error.

Information derived from the analysis of episodes and phases formed part of a larger analysis that characterized subjects' protocols in terms of overall task accomplishment (outcome variables), approaches to problem solving (method variables), and specific solution-related factors (solution variables). Both qualitative and quantitative variables were examined.

Four binary variables resulting from this analysis served as input to a macro-analysis that identified strategic decisions the subjects made in debugging. These four decisions represent significant choices subjects faced, either explicitly or implicitly, in debugging a program. The binary decisions, in the sequence in which subjects considered them, are:

1. Whether subjects examined the program or the output first.

2. Whether subjects engaged in active or passive examination of the problem

3. Whether subjects were constrained by the hypotheses they stated.

4. Whether subjects developed a model of the program structure and deduced a causal representation of the error in terms of that model.

Subjects were then characterized according to the strategic decisions they made. Figure 3 shows a decision tree representation of the strategy paths followed by the subjects. With 4 binary decisions, there are a total of 16 possible paths. The number of subjects choosing each path is shown on the diagram. Subjects followed 6 of the 16 paths. The expert and novice programmers according to the *ex post* classification were then compared with the groups of programmers following certain strategic paths derived from the process analysis. This comparison permitted the debugging strategies used by those programmers classified as experts and those classified as novices in this study to be identified. Experts and novices each followed 3 paths (strategies 2, 4, and 6, and 1, 3, and 5, respectively).

Table 5 shows a decision table representation of the decision tree presented in Figure 3. Note that two factors determined expert behavior in this diagnostic task: the ability to pursue a breadth-first search for the error (Nilsson, 1980; Feltovich, 1981) and the ability to think in systems terms (Johnson, Hassebrock, Duran, and Moller, 1982). Programmers who were constrained by the hypotheses they generated were novices. Moreover, programmers who engaged in breadth-first search for the error but who did not formulate a model of the program structure and conceive of the error within that context were likely to make mistakes. They were therefore regarded as novices. Whether subjects initially examined the output of the program had no effect on problem solving, neither did reading modules versus mentally execut-

Table 3

Debugging Performance Measures

Measurement Variable	Direction
Debugging Effectiveness	
Debug time adjusted for verbalization rate	N > E for L
Number of mistakes made	N > E for L
Debugging Efficiency	
Number of changes in debugging functions	N > E for L
Number of reversals to the Debug Program function	N > E for L
Number of changes in program location	N > E for L

Table 4

Summary of the Significant Results for Hypotheses Testing the Effects of the *ex post* Classification on Debugging Function and Debugging Activity Variables

Dependent Variable	*Main Effects* Significance Direction	*Interaction Effects* Significance Direction
Debugging Functions		
HF8: Debug Program (DB)	.008** N > E	
HF9: Find Problem (PF)	.015* E > N	
HF1: Generate Hypothesis (HFG)		.046*[1]
HF3: Repair Error (EAR)	.025* N > E	
Planning Activities		
HAP2: Evaluates a goal (GOE)	.029* N > E	
HAP5: Indicates goal not satisfied (GON)	.043* N > E	
Knowledge Building Activities		
HAK3: Gathers information from statement (IGS)		.028* N > E for L L > H for E
HAK6: Evaluates information (IGE)	.030* E > N	
Bug-Related Activities		
HAB8: Clue (CL)	.040* E > N	
HAB1: Finds a clue (CLF)	.004** E > N	
HAB6: Makes error (ERM)	.003** N > E	

[1]The significant interaction effect for the Generate Hypothesis function (HFG) did not show significant treatments using the Newman-Keuls test.

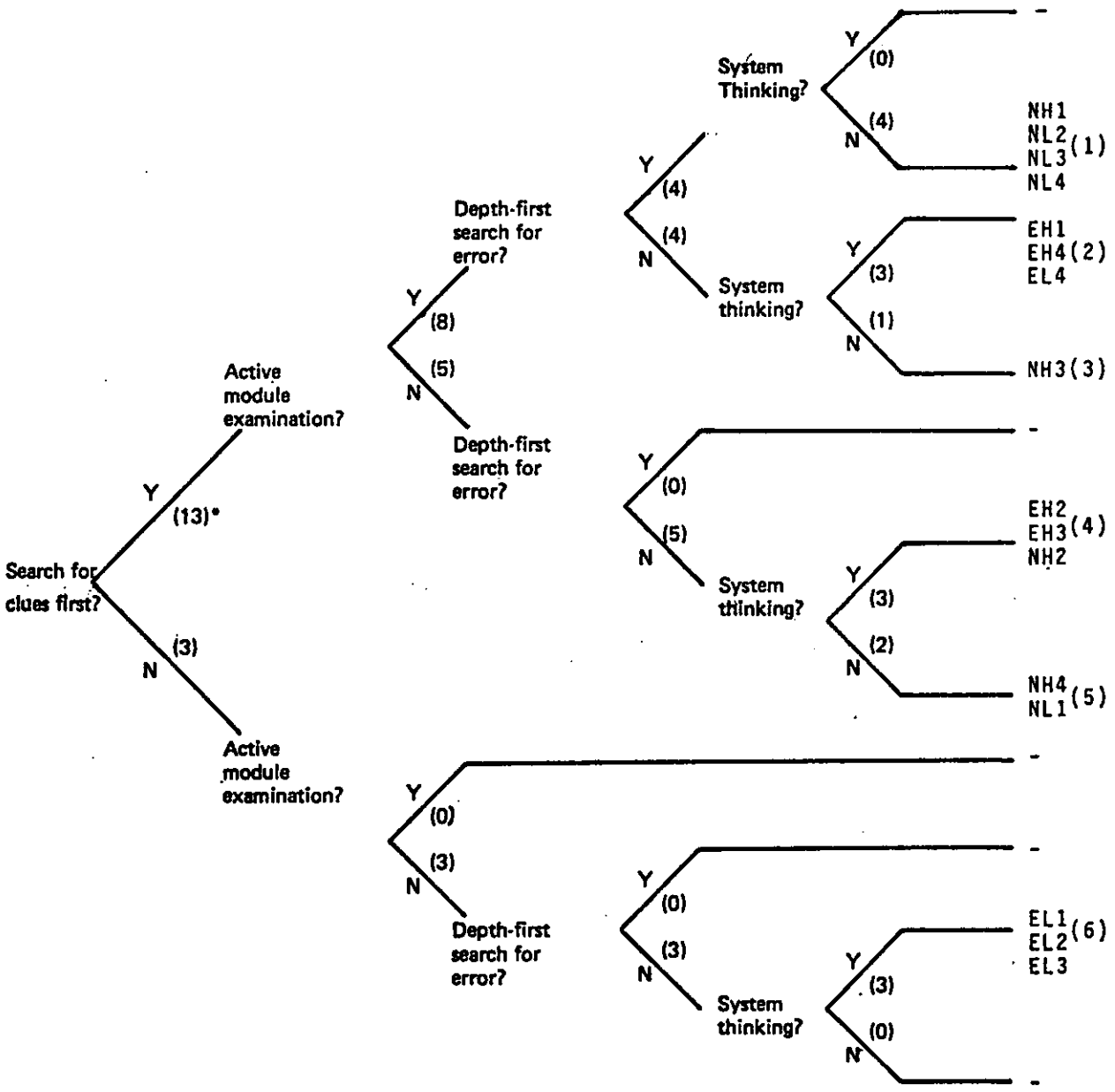

Figure 3

Strategy Paths Followed by Programming Subjects

*The numbers in brackets on the branches represent the number of subjects following that strategy.

[1]The alternative to searching first for clues to the problem is to examine the program structure and function and then to search for clues.

[2]Active module examination is distinguished by:
(a) initially following the execution path of the program rather than the lexical sequence;
or
(b) actively searching for the error rather than first understanding the program.

[3]All subjects who were not recorded as being constrained by their hypotheses were regarded as engaging in breadth-first search for the error.

ing modules. The decision table, based on only two binary conditions, classified 15 of the 16 programmer subjects in the same way as the *ex post* skill classification, which was based on the chunking ability of the subjects.

Discussion and Conclusions

Two principal aspects of this study addressed the issue of what constitutes skill in debugging computer programs. The first aspect, from the viewpoint of the conduct of the study, was the nature of the classification used to derive groups of expert and novice subjects for the study. The *ex post* classification more effectively described performance than the *ex ante* or manager classification. Hence, these results support the concept on which the *ex post* classification was based, viz., that subjects' problem solving processes result in significant variability in performance that is difficult to capture except by explicit recognition of those processes. Hence, the classification itself provided information on certain differences in expert and novice debugging processes. Expert debuggers in this study were those who could more effectively chunk programs. They therefore exhibited disciplined approaches to problem solving, pursing similar types of behavior rather than frequently changing mode of behavior, checking on the clues to the problem, and changing reference points within the program.

Support for the use of chunking abilty as a measure of debugging expertise was provided by the analysis of subjects' strategy paths. Except for subject NH2, classification of subjects according to their high-level problem solving capabilities and their approach to modeling the system resulted in the same programmer classification as that based on chunking abilty. Hence, a micro-analysis of debugging activities and a macro-analysis of debugging strategies produced simlar results.

The major aspect of this study that addressed debugging expertise was the notion of the importance of situation-dependency in problem solving. Both the content and the process analyses assessed the extent to which experts and novices responded to factors in the task environment as opposed to following an internal model of the debugging process. The content analysis demonstrated that the situation-dependent approach (debugging activities) was capable of eliciting more information than the model-based approach (debugging functions). This was due partly to the greater level of detail in the categories examined but also to the nature of a model, which permitted capture of only certain selected aspects of a process. For example, the increased error repairing behavior of novices was captured by both functions and activities (EAR and ERM) as was the problem formulating behavior (PF and CLF). However, gathering information from statements activity (IGS) showed significant effects, while no significant effects were observed for the relevant functions (CR, CRR, CRL, and CRP). The level of description in the model-based approach was apparently too global to detect the more subtle differences in information gathering behavior. The significant result for information evaluating activity (IGE), on the other hand, could not have been captured by the model-based approach as there was no "evaluate" function. Furthermore, the fact that experts spent more time than novices in the Find Problem function (see Table 4) is evidence of greater situation-dependency on the part of experts.

The process analysis illustrated clearly that novices did not respond to the situation as it revealed itself. This was particularly evident in the four criteria used to conduct the analysis of subjects' debugging strategies: the sequence of examining the program and the problem, the extent of active involvement in determining program functioning and/or the module in error, the attachment to stated hypotheses, and the ability to create a model of normal program functioning. The first three factors illustrated directly that expert debuggers approached the problem in a more relaxed manner than novices, i.e., they responded to the task environment rather than to an internal model of the debugging process. The only (3) subjects who examined the program before the output listings were experts. Experts tended to read through the program in lexical rather than execution sequence. Experts did not appear to be guided by any hypotheses they stated, let alone addicted to them as were novices. The fourth factor, formulating a model of correct system operation and perceiving the effects of the program error in terms of that model, is a consequence of the high-level, situation-dependent problem solving approach of experts.

Evaluation of these four factors in terms of the strategy path analysis revealed, however, that only two of these factors have significant consequences for problem solving expertise. Being constrained by the hypotheses they stated and therefore engaging in a depth-first search for the error had significant (negative) consequences for expertise. So too did the lack of ability to conceptualize the program and the error in it as a system. Hence, expertise is associated with two data-driven or situation-dependent characteristics: breadth-first search for the error and the ability to create a model of normal program functioning.

Results obtained by examining debugging expertise lead to the following strategic propositions that can be tested in future research:

1. (a) Experts use breadth-first approaches to problem solving and, at the same time, adopt a system view of the problem area.

Table 5

Decision Table for Determining Expert and Novice Subjects According to the *ex post* Programmer Classification

		Rules		
		1	2	3
Conditions	1. Breadth-first search for error	Y	Y	N
	2. System Thinking	Y	N	-
Actions	A. Designate Expert	X		
	B. Designate Novice		X	X

[1]This table approaches the designation of experts and novices from the viewpoint of experts as opposed to Figure 3, which approached it from the viewpoint of novices. Figure 3 derived from the analysis in this chapter which identified constrained problem solving as a characteristic of novices, while a more positive approach identifies the characteristics of experts.

(b) Experts are proficient at chunking programs and hence display smooth-flowing approaches to problem solving.

2. (a) Novices use breadth-first approaches to problem solving but are deficient in their abilty to think in system terms.

(b) Novices use depth-first approaches to problem solving.

(c) Novices are less proficient at chunking programs and hence display erratic approaches to problem solving.

Further investigation will serve to extend and refine the theory and also to set boundaries on the applicability of the strategic propositions.

REFERENCES

Atwood, M.E. and Ramsey, H.R., "Cognitive Structures in the Comprehension and Memory of Computer Programs: An Investigation of Computer Program Debugging", *NTIS*, AD-A060 522/0, 1978.

Bouwman, M.J., "Financial Diagnosis: A Cognitive Model of the Processes Involved", Unpublished Doctoral Dissertation, Carnegie-Mellon University, 1978.

Brooks, R.E., "Studying Programmer Behavior Experimentally: The Problems of Proper Methodology", *Communications of the ACM*, Volume 23, April, 1980, pp. 207–213.

Dreyfus, S.E., "Formal Models versus Human Situational Understanding: Inherent Limitations on the Modeling of Business Expertise", *Office: Technology and People*, Volume 1, August, 1982, pp. 133–165.