Leventhal, Laura Marie

PERCEPTION OF SOFTWARE QUALITY: AESTHETICS AND EXPERTISE

The University of Michigan PH.D. 1987

PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark __✓__.

1. Glossy photographs or pages _____
2. Colored illustrations, paper or print _____
3. Photographs with dark background _____
4. Illustrations are poor copy _____
5. Pages with black marks, not original copy _____
6. Print shows through as there is text on both sides of page _____
7. Indistinct, broken or small print on several pages ✓
8. Print exceeds margin requirements _____
9. Tightly bound copy with print lost in spine _____
10. Computer printout pages with indistinct print _____
11. Page(s) _____ lacking when material received, and not available from school or author.
12. Page(s) _____ seem to be missing in numbering only as text follows.
13. Two pages numbered _____. Text follows.
14. Curling and wrinkled pages _____
15. Dissertation contains pages with print at a slant, filmed as received ✓
16. Other _____

_____

_____

University
Microfilms
International

PERCEPTION OF SOFTWARE QUALITY: AESTHETICS AND EXPERTISE

by

Laura Marie Leventhal

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer and Communication Sciences)
in The University of Michigan
1987

Doctoral Committee:

Professor Stephen Kaplan, Chairman
Associate Professor Larry K. Flanigan
Professor John Holland
Professor Rachel Kaplan
Assistant Professor Terry Weymouth

For Brian

ACKNOWLEDGMENTS

Of the many people who contributed to aspects of this dissertation, I would like to specifically acknowledge the contributions of a few. My family has been a bottomless well of support. My husband, Alan Jaffee has undoubtedly given up a piece of his life in the preparation of this document. My mother, Zella Jones Leventhal, has provided not only support but practical assistence as well, by baby and dogsitting. My son, Brian continues to remind me that things, not related to work and dissertations, deserve attention and provide joy.

I have had the privilege of participating in the "Friday Seminar." The members of this group have generously shared their insights with me and been willing to listen to my ideas as well.

The members of my committee have helped me in many ways. Larry Flanigan is, without a doubt, the best teacher that I had while I was a graduate student. I appreciate his comments which relate this work to teaching. John Holland's original ideas, particularly at the beginning of this project, were thought-provoking and helpful. Terry Weymouth seemed to be especially sensitive to what this process has been like for me. I have special thanks for Rachel Kaplan for her generous participation on my committee at a late date and also for providing the starting point for this work with her designers and "clients" study.

Thanks also to Larry Dunning and John Townsend for the opportunity to collect data from their students and to the numerous students who voluntarily participated in the study. Thanks to Janet Talbot for her critical assistance in the data analysis in this study.

Most especially however, I feel gratitude to two special people. My association with Steve Kaplan has been one of the best things to come out of my graduate experiences. Not only has he shared his ideas, which form a basis for this work, but he has freely shared his time and insights as well. His interest in both my professional and personal well-being is greatly apprecicated.

My father, Donald Leventhal, is not here to see the completion of this dissertation or my graduate work. However I have felt throughout this long process a strong connection with the innumerable things which I learned from him. In particular, I have often thought of him, searching through the rubble left by a tornado, for his dissertation which had been already been completed for many years.

TABLE OF CONTENTS

DEDICATION ii

ACKNOWLEDGMENTS iii

LIST OF FIGURES vii

LIST OF TABLES viii

LIST OF APPENDICES ix

CHAPTER

I. AESTHETICS AND EXPERTISE:
TWO CENTRAL ISSUES IN COMPUTER SCIENCE 1

Empirical Study of the Issues: A Necessity
Statement of the Problem
Some Outcomes of this Study
Summary

II. AESTHETICS AND EXPERTISE: A THEORETICAL MODEL 6

Research in Expertise: The Role of Problem Representation
Environmental Preference: An Informational Theory
Appropriate for Computer Science
Problem Representation and Preference: What is the
Connection?
Summary - Problem Representations and Preference
Notes to Chapter II

III. DESCRIPTION OF METHODOLOGY AND TEST INSTRUMENT 21

Description of Chosen Methodology
Selection of Experimental Unit
Selection of Content of the Test Instrument
Description of the Data Collection Procedure
Description of the Problems and Solutions
Summary of Methodology, Test Instrument, and Procedure
Notes to Chapter III

IV. PREDICTION OF AESTHETICS IN EXPERTS AND NOVICES. 39

Analyses
Results
Preference and Problem Representations
Summary
Notes to Chapter IV

V. PERCEPTUAL CATEGORIES: EXPERTS. 48

Expert Factor Structure
Relation of Categories to Preference
Expert Categories and Environmental Preference Framework
Summary
Notes to Chapter V

VI. PERCEPTUAL CATEGORIES: NOVICES. 62

Novice Factor Structure
Relation of Categories to Preference
Novice Categories and the Environmental Preference Framework
Summary
Notes to Chapter VI

VII. REVIEW, CAUTIOUS APPLICATIONS, AND CONCLUSIONS. . . . 75

Review of Results
Review of Themes
Some Cautious Applications
Conclusion

APPENDICES 84

BIBLIOGRAPHY 133

LIST OF FIGURES

Figure

4.1. Prediction Framework for independent variables 41

4.2 Partial Correlations for Experts and Novices using prediction framework from Figure 4.1 43

5.1 Preference Framework for Experts. 59

6.1 Preference Framework for Novices. 72

A.1 Similarities and Differences between Current Study and Soloway and Ehrlich (1984) 89

LIST OF TABLES

Table

2.1. Kaplans' Model of Environmental Preference. 14
3.1 Construct Affordances and Violations 27
3.2 Matrix Rotate Items 32
3.3 Sorting Items 32
3.4 Main Procedure Items 34
3.5 Numerical Calculation Items 36
3.6 Searching Items 36
5.1 Expert Factor Characteristics 50
5.2 Mean Preference Levels for Expert Categories 54
6.1 Novice Factor Characteristics 63
6.2 Mean Preference Levels for Novice Categories. 67
A.1 Details of Discourse Rule Violations 87

LIST OF APPENDICES

Appendix

A. Unsuccessful Analyses 85
B. Sample Task Item ..... .90
C. Sample Characteristics 125
D. Derivation of Prediction Framework 128
E. Facsimile of Data Used in Analysis of Covariance ..... .129
F. Analysis of Covariance for Two Subject Groups ..... .131
G. Expert and Novice Factor Loadings 132

CHAPTER I

AESTHETICS AND EXPERTISE: TWO CENTRAL ISSUES IN COMPUTER SCIENCE

That expertise is a relevant theoretical and practical issue in computer science and related domains is no surprise. Experts write software systems and manuals for use by non-experts. Experts provide both the leadership and the materials for the education of non-experts. Experts even develop software systems, like word processors, so that the performance of the novice may approximate that of an expert.

That aesthetics is a useful notion is certainly no surprise to practitioners in computer science either. Within the discipline there appears to be widespread agreement that computer-related products, particularly software systems often have an "artlike" quality. (e.g., Bentley, 1986). Aesthetic issues appear to span both computer science education (e.g., a popular introductory text by Savitch, 1984, is entitled PASCAL: An Introduction to the Art and Science of Programming) and practice (e.g., Bentley's "Programming Pearls" column in the widely-read Communications of the ACM is devoted to issues of elegance in software).

The interaction between expertise and aesthetics is an especially interesting and relevant issue for computer scientists. For instance, since computer science experts are responsible in large measure for so many types of computer systems, their preferences, of course, have the potential to have a tremendous influence on both products experienced by end-users and the computer-product marketplace. Consider:

> In 1975, Steve Jobs and Steve Wozniak started the Apple Computer Company in Jobs' parents' garage. By 1985,

> the company had grown to a 2 billion dollar company. Then Jobs was forced out of the company.
>
> Among the descriptions of Jobs' fall , one of the most striking involves decisions for the design of the Apple MacIntosh. The MacIntosh was designed to be a business computer. Jobs however refused to put fans into the MacIntosh or any other Apple computer; yet, fans are required to support the storage demands of an office computer. The reason - fans, in Jobs' opinion resulted in *an inelegant design.* (my italics; Gelman, et al., 1985)

It seems clear that aesthetic considerations permeate much of the work of computer science experts. To learn more about the interaction of these two issues would appear to have a special urgency for the practice of computer science.

Empirical Study of the Issues: A Necessity

Aesthetics and aesthetics-related terms have entered into the everyday language of computer science, but the nature of such an aesthetic remains virtually unexplored. As in the more general realm of the human-computer interface (cf. Shneiderman, 1979; Sengler, 1983; Hagglund and Tibell, 1983; Arblaster, 1983; Maass, 1983), what one finds instead are many examples of unsubstantiated guidelines (cf. Cooper and Clancy, 1982) and folklorish beliefs. For example, a common belief in the popular folklore of computer science is that the style of each computer science expert is an individual signature.

However, within the discipline, several authors have called for a different approach to issues of the human-computer interface. Under this alternative, models of behavior, based on empirical research, would be widely-familiar and available for practioners (cf. Moran, 1981). A move toward this approach seems to be an absolute necessity within computer science. As an example, these issues could be incorporated into software engineering courses. A recent survey of content in undergraduate software engineering courses indicated that human interface issues are perceived by software engineering instructors to be a focal point in a typical course (Leventhal and Mynatt, in press) Unfortunately, few

software engineering texts address these issues at all! It would appear that computer science educators are relying on materials of questionable quality or validity in the training of future computer scientists. This apparently ad hoc treatment of human issues is potentially devastating if in fact this is the type of information which most professionals are acquainted with.

Of all the human interface issues, none seems to be more folklorish than that of aesthetics, particularly the aesthetics of computer science experts. However, as researchers in a variety of other disciplines have demonstrated, aesthetic issues can be studied empirically. In particular, as this study will demonstrate, it is possible to cast issues of the nature of aesthetics for both computer science experts and novices into the framework of empirical research.

As in any study of human-computer issues, the investigation of aesthetic issues is made more difficult by the nature of the research environment. On the one hand, the computer scientists who are most familiar with the issues in a practical sense are typically unprepared by their training to conduct meaningful experiments. Numerous critiques of empirical work in the field find the work characterized by misuse of statistical tools, use of inappropriate stimuli, use of inappropriately small samples, and use of inappropriate subject groups (cf. Sheil, 1981; Brooks, 1980; Moher and Schneider, 1981, 1982; Sayward, 1984; Cioch, 1985). By contrast, trained experimenters, such as experimental psychologists often have only a surface level understanding of the scope of or importance of issues in computer science. These difficulties suggest that a reasonable study of behvioral issues, such as aesthetics and expertise, must be interdisciplinary in nature. Studies of this type must at once preserve a rigorous methodological approach and a sensitivity to the relevant computer science variables.

Statement of the Problem

The purpose of this study is to explore the relationships between aesthetic issues and expertise levels in a rigorous, experimental fashion. The purpose of the study is twofold: 1. To identify patterns of aesthetic preference in a computer

science setting and 2. To investigate the interaction between these patterns of preference and increasing levels of expertise in computer science.

Since the role of aesthetics and its interaction with computer science expertise remains virtually unexplored, there are few precedents on which to base a study of these issues. Two issues are particularly challenging in this setting: 1. Identification of an appropriate theoretical framework in which to base such a study and 2. Selection of an appropriate methodology and experimental materials.

Identification of an appropriate theoretical framework has been hindered by the folklorish nature of computer science aesthetics and expertise. One is forced instead to consider theoretical models which have been posed outside of the discipline. In Chapter II, a model of expertise and a model of aesthetics and preference are discussed separately. Some potential interactions, consistent with both theories and some experimental work, are then described.

Selection of an appropriate methodology and experimental materials is also made difficult given the lack of relevant precedents in this area. One is forced to redefine methodologies from outside computer science into uniquely computer science forms. Chapter III describes both the selected methodology and form of the stimulus.

Some Outcomes of this Study

It is clear that within the folklore of computer science, aesthetics, particularly in the hands of experts is a real and relevant issue. In practice, experts play vital roles in the education of and communication with non-experts, as well as being the principal providers of software products. Yet even experts themselves are often unaware of the ways in which they differ from novices. (Kaplan, 1977; Kaplan and Kaplan, 1982) Identifying differences between the groups has far-reaching implications for both the development of new experts and the interactions between novices and experts.

A second, noteworthy consequence of a study of this type relates to a constant yet implicit theme throughout this document. In the experience of this

author and others, (cf. Molzberger, 1984) computer science professionals consider themselves to be artists. To someone outside of the field this is likely to seem to be a ludicrous idea, given the apparently rigorous and rational nature of the tasks in the discipline. Identifying measurable components of aesthetics in computer science is a sort of justification and confirmation for professionals in the domain that they are more than just technocrats.

Summary

In any field, high quality work is marked by a concern for aesthetics (cf. Hamming,1981). Aesthetic considerations distinguish elegant work from that which is merely adequate. In computer science, aesthetic judgements have not often been acknowledged by experimenters. That practitioners know differently is demonstrated in both the folklore of the discipline and in the emphasis in computer science education on issues of style. Experts under the influence of aesthetics often make decisions with far-reaching consequences.

This study is a first step towards the development of a model of computer science aesthetics and the interaction with expertise level. Such a model is potentially applicable in many ways within the discipline. This opportunity to unite empirical research and practice offers the exciting potential both to extend general understanding of the human-computer interface and to enhance experts' understanding of themselves.

CHAPTER II

AESTHETICS AND EXPERTISE: A THEORETICAL MODEL

What constitutes an appropriate study of aesthetics and expertise in computer science? Before trying to combine the themes of expertise and aesthetics in the context of computer science, it may be useful to explore them separately. Since work in these domains is sparse within computer science, one must turn elsewhere for theoretical models and then determine their applicability to the present situation.

Research in Expertise: The Role of Problem Representation

One of the strongest themes to emerge from expertise research is that experts focus much of their problem solving activity on the formation of an adequate problem representation. Several authors have suggested that the quality of the problem representation is a determining factor in the ease of problem solving (cf. Hayes and Simon, 1976; Newell and Simon, 1972) A problem representation is an internal model of the problem; it may include information about the problem itself, alternate formulations of the problem, problem constraints, solution strategies, and solution information.

Formation of a complex representation requires two types of activities by the expert. The expert must be able both to recognize salient environmental information and to refine the problem representation until it is adequate.

Recognizing Environmental Patterns

The expert must be able to recognize the salient environmental patterns in

order to build a useful problem representation. Many studies of expertise indicate that with increasing experience experts develop a powerful perceptual facility. De Groot's early work in this area examined expertise in chess. It provided the technique for the first experimental studies of computer science experts (cf. Shneiderman, 1976; McKeithen, Reitman, Reuter, and Hirtle, 1981; Egan and Schwartz, 1979). In the computer science version of this methodology, the subject is typically presented with programs for a short time period. The program statements are arranged in either meaningful or random order. After the presentation, the subject is asked to reconstruct the program statements. The computer science experts show superior performance when presented with statements in meaningful order; the performance by the expert is degraded to the level of the novice in the random order condition.

The results of these studies indicate that computer science experts rely, at least in part, on highly-developed pattern recognition processes, rather than on logically-guided retrieval processes. The facility of experts to process meaningful information during the short duration of the experimental task is highly suggestive of this conclusion.

In recognizing salient patterns, the expert is able to ignore distracting, irrelevant information. This was demonstrated by Soloway, Ehrlich, and Bonar (1982) who asked experts and novices to fill in a missing line of code in presented programs. Some of the programs contained a superfluous line of code. The performance of the novices was degraded in the presence of noise; the performance of the experts was unaffected. This result suggested that the experts selected the salient stimuli and completed the task. The line of noise presumably was not included in the resulting expert problem representations.

Refining the Problem Representation

The expert must also be able to manipulate the problem representation until it reaches an appropriate form. Jeffries, et al. (1981) traced the development of a software system from problem statement to a solution. The primary focus of

this research was the process of development of problem representations. Models of the experts were found to incorporate more global constraints and alternatives than those of the less expert. These constraints and alternatives affected the way that the experts interpreted the problem.

The experts in this study used a "top-down" strategy to decompose the problem into parts. This strategy was repeated on problem subparts several times before program code was actually generated. In contrast, the advanced novices were able to break the problem into meaningful subunits, but were unable to repeatedly decompose the subunits. The novices developed an outline of the steps for the solution and then developed program code. The prenovices simply generated program code. These results suggest that the experts developed sophisticated problem representations before they generated a solution. One can infer that the top-down decompostion strategy allowed the experts to "fill in" the detailed constraints in the problem. The novices and pre-novices, lacking a rich representation, had no choice but to develop a solution directly from the problem statement.

Voss , Tyler, and Yengo (1983) reported similar results from their collected verbal protocols of novices and experts solving a political science problem. Their problem was typical of political science problems; it was open-ended, with no single correct answer.

Like the computer science experts in the Jeffries study, political scientists appeared to spend a great deal of time forming a problem representation when they were solving a problem in their area of specialization. These experts developed constraints for the problem. These constraints were then used to bound the problem. Not surprisingly, the representations of these experts were heavily influenced by the constraints of the problem. The experts proposed solutions which were more abstract than the solutions of novices.

Since the solutions to the problem were never clearly right or wrong, the experts spent much of their time generating rationales for their solution. Their arguments were attempts to show that their solution was both workable and

feasible.

In contrast, the novices did not focus on the construction of a problem representation. The problem representations reflected little consideration of constraints. The novices showed little argument development and the arguments relied on general knowledge of logic and psychology to justify claims. The solutions which were generated were simpler than those of the experts.

The experts in the Voss, et al. study resembled the computer science experts in their focus on the formation of a problem representation. However, unlike the protocols of computer science experts, the protocols of political scientists showed extensive argument development. Voss and his associates attributed this emphasis on argument to the open-ended nature of the problem, rather than to differences between political science and other domains. This conclusion seems reasonable, especially in light of the apparently universal focus on problem representation among experts across numerous domains

Larkin's (1983) collected verbal protocols from physics experts are further indication that experts may refine their problem representations extensively before finding a final solution. Her protocols indicated that experts often reject potential hypotheses before the hypotheses are fully developed into a final form. She reported that physics experts, in solving medium-difficulty problems, selected the entities of their preliminary problem representation quickly. The first representation was replaced by another when it was found to be unacceptable.

Once experts have formed an adequate representation, a specific problem solution usually follows in a straightforward way. Larkin (1983) reported that experts who were solving relatively difficult problems generated the proper physics solution (equations) only after they had developed complex representations of the problems.

Components of Expert Problem Representations

Not only does the complexity of problem representations appear to increase with increasing expertise, but the entities in expert problem

representations seem to differ from those of novices. In particular, the components become increasingly abstract. For example, Kahney (1983) used two functionally equivalent recursive programs which were syntactically different. Both of the programs satisfied the stated requirements in the experiment. Experts were able to recognize that both programs were satisfactory; novices were only able to identify the obvious solution. These results suggests that the experts focused on the abstract functional properties of the programs rather than the syntactic level details.

Adelson (1984) examined the interaction of abstractness and expertise by using a comprehension task. Expert programmers were better able to answer questions about the function of programs; novices dealt more effectively with questions about the specific ways that the programs worked. Experts who were provided with a program and a description of how the program worked were able to answer both abstract and concrete questions with no significant differences. When delays were introduced between the presentation of the program and description, and the comprehension task, experts again performed better on the abstract questions than on the concrete questions. The novices, on the other hand, performed better with the concrete questions.

Several other studies in computer science, physics, and mathematics suggest that increasing expertises is accompanied by decreasing specificity (Weiser and Shertz, 1983, Soloway, Ehrlich, Bonar, 1982, Chi, Feltovich, and Glaser, 1981, and Lewis, 1981).

Summary: Expertise and Problem Representation

Much of experts' superior problem solving ability, within their domain of expertise, appears to be related to their facility to form complex problem representations. From the formation of the initial problem representation to the generation of a solution, the expert benefits from a more adequate problem representation. In the formation of the initial problem representation, the expert relies on enhanced perception; the mechanism is fast, automatic, unconscious,

and relatively accurate (Kaplan, 1978).

During the extended refinement period, experts reject incomplete and inappropriate hypotheses, and generate both solutions and rationales. Because experts are able to reject incompletely formed hypotheses, they can consider more alternatives than novices who work with only one hypothesis. Once an appropriate hypothesis is considered in detail, the specific solution usually follows in a straightforward way. Generating rationales in the process, the expert is prepared for the future when he or she may have to defend the proposed solution.

The transition with expertise toward more abstractness has an advantage as well. While the entities in the problem representation are increasingly incomplete, they are, within a limited set of situations, increasingly general purpose. So, the expert is able to solve a class of problems, which on the surface seem to be different, with equal ease.

Environmental Preference: An Informational Theory Appropriate for Computer Science

The general lack of studies of aesthetics in computer science is paralleled by the lack of a theory of computer science aesthetics. One area of psychological theory which is particularly relevant to aesthetics in a computer science setting involves a theory of preference in the physical environment. The physical environment and the computer science environment have many surprising similarities; the information processing demands of software engineering tasks and the physical environment are remarkably similar. Consider that in both cases, the setting is large and complex; consequently the entire setting can not be experienced at one time. Both the computer setting and the physical environment can, at once, present too much and too little information. The physical environment and the interfaces appear to be irregular. But patterns occur in both cases. It is possible to become lost in either case; finding a "path" in either domain can be considered to be a potentially

challenging wayfinding activity.

The notion that an environmentally oriented theory of human information processing is an appropriate framework from which to study the human-computer interfaces may seem, at first to be a little farfetched. An interface task has traditionally been considered to be a verbal task, whereas functioning in an environment has been considered to require more spatial processing. However, several authors have indicated that the distinction, based on content, between these two orientations may not be so clear cut. Huttenlocher (1976) presents an arguement that much of language use involves spatial processing, particularly in the evaluation of relations. Also, Luria's (1973) patients with brain lesions in the region of spatial processing exhibited difficulty in processing sentences with relationships or complex structure. Nelson and Castano's (1984) more recent work shows little difference in the processing and use of pictures and words.

A central characteristic of the human-environmental interface is that people do not experience the environment as neutral. To function in a physical world, a person must be able to evaluate alternative environments. Environments can exhibit particular types of patterns which are more or less favorable to information processing demands. Not surprisingly, people make preference judgements quickly and without conscious awareness (Zajonc, 1980).

While several models of environmental preference have surfaced in the context of landscape assessment (Daniel and Vining, 1983), only one has approached the environment in informational terms (Kaplan and Kaplan, 1982). In particular, it relates the organization of information to preference. In their model, preference has two primary indicators:

How engaging the environment is
How easily the environment makes sense.

An environment that is engaging offers an opportunity to learn more about the setting. An environment that makes sense is recognizable and predictable;

the pieces in the scene readily "fit together." Making sense and engageement do not represent the two boundaries of a single continuum. Rather a prefered environment exhibits both characteristics (Kaplan and Kaplan, 1982). Evaluation occurs not only in the present context, but also with respect to the future. Table 2.1 describes how both making sense and engagement deal with this time dimension.

Within the Kaplan and Kaplan framework, then, a preferred environment is one that is high in both engagement and making sense. The theory has shown useful predictive power in a variety of physical environments and environmental design settings (cf. Kaplan, 1987). It also has a remarkably reasonable appeal. Consider the sorts of pressures which are applied by the environment. In order to find one's way, one should prefer a setting where the pieces fit together in a meaningful pattern. Yet environments are always different. The more that one can learn about environments, the more prepared one is for future environments.

The two-factor environmental preference model has been successfully applied to phenomena outside of environmental psychology. Herzog and Larwin (unpubl.), for example, have recently found that making sense and engagement are important variables in preference for captioned cartoons.

Various other studies outside of environmental design have demonstrated relations between informational characteristics of the stimulus and aesthetics and preference, although the authors have not interpreted the results using the Kaplans' framework. In particular, results which relate preference in art (Nicki, 1981; Child, 1981), graphics (Walker, 1981), narratives (Moynihan and Mehrabin, 1981), music (Crozier, 1981), and experimental aesthetics (Hare, 1981) to informational variables have been reported.

Several authors have suggested that an environmentally-based model of preference is appropriate for computer science settings. In particular, Weiser (1979) has speculated that the two-factor, engagement-making sense model is a reasonable one. In a study of intrinsic motivation, Malone (1981) described

MAKING SENSE	PRESENT	I understand what is happening in this setting
	FUTURE	Given what I can see of the whole setting,I know that I probably will not become lost
ENGAGEMENT	PRESENT	This setting has enough information to support an internal model
	FUTURE	I could find interesting information in this setting if I explored it further

based on Kaplan and Kaplan (1982)

Table 2.1 - Kaplans' Model of Environmental Preference

variables which contributed to preference in computer games. He suggested that informational complexity, which is an aspect of engagement, has an effect on preference.

Molzberger (1983, 1984) described interviews with very expert programmers whom he called "superprogrammers". The results of the interviews also fit into at least one aspect of the Kaplans' preference framework. In particular, he found that these software experts stressed the role of harmony and wholeness in aesthetically pleasing programs. Logic errors were experienced as disruptions in aesthetic harmony. Since harmony and wholeness typically imply that pieces fit together, these experts appeared to be describing an aspect of the making sense dimension of the Kaplans' model.

The two-factor environmental model has an intuitive appeal as a framework for computer science preference as well. Consider Kernighan and Plauger's (1979) Elements of Style. The book was an attempt to concretize some of the stylistic variables that affect programming. While most of the rules have never been tested empirically, this book remains a standard reference text among computer scientists. The major premise in the text is that good style leads to good programming. Nineteen of the sixty-one rules can be considered to be aesthetic rules. These nineteen rules focus on one central idea: clarity. This notion of clarity is expressed with terms like "uniformity", "simple", "easy", avoid "ambiguity", and avoid "confusion". In this respect, one can surmise that Kernighan and Plauger were attempting to express much the same idea as the Kaplans' making sense dimension.

Several studies, while not specifically addressing issues of aesthetics reveal that the components of informational structure, similar to the two-factor preference matrix, relate to problem solving issues outside of preference. Most notably in computer science, Soloway and Ehrlich (1984) investigated the consequences of rules of programming discourse. Violation of these rules, according to Soloway, Ehrlich, and Black (1983) would horrify an experienced programmer by being in conflict with their expectations. Soloway and Ehrlich

found that expertise level, violation of the rules, and the interaction of the two issues had a significant effect on program comprehension, as measured by performance. One can infer that violation of these expected conventions had a negative effect on degree to which the programs involved made sense.

Similarly, Black and his colleagues have demonstrated that coherence relations affected the inferences that people make in the process of story understanding. In particular, sentences which are related causally, by setting, motivation, common referents, or consistent points of view tend to have higher levels of coherence and increased comprehension, as demonstrated by memory tasks. Again, coherence, in his presentation, can be considered to be related to the making sense dimension of the environmental preference model (Black and Bern,1981; Black, Galambos, and Read, 1984, Black, 1984).

It is worth noting at this point, that while the literature in computer science and related fields seem to fit, at least indirectly, into the Kaplans' framework, there appears generally to be an imbalance in these reports. The theme of making sense is clearly figural in this literature. The notion that a variable of engagement is also relevant has received considerably less emphasis.

Problem Representations and Preference: What is the Connection?

The two theoretical issues of problem representation and preference, seem to be unrelated. Problem representation is related to a cognitive entity which becomes increasingly important with increasing expertise. The facility to develop an adequate problem representation is related to the powerful problem solving abilities which are developed by experts. Preference on the other hand refers to an affective type of coding for patterns of information, which are manifest as environmental scenes, works of art, narratives, musical compositions, or as computer science tasks. Informational patterns with high preference at once facilitate structuring and provide enough diversity to sustain the activity.

Surprisingly, patterns which result in high preference also have a high payoff for the formation of a problem representation. Consider a person in a

problem solving setting. The formation of an initial representation is enhanced if the setting makes sense on the surface since the person can distinguish the elements of the problem and their interrelations. A problem which is engaging on the surface holds promise. It suggests future discovery if the person continues to work on the problem; that is, refinement of the problem representation may be rewarded in terms of the new information and the interpretation that it yields.

Similarly, a problem which makes sense suggests that the entities of the problem will fit together as the problem representation is refined. The solver is not likely to encounter chaos in the problem as the refinement process continues. If the problem is engaging it presumably will continue to provide opportunities for learning. This promise encourages the solver to continue to refine the problem representation toward a solution.

Walker (1981) reviewed empirical evidence relating problem representations and preference. He suggested that after repeated experience with a given event, that event will undergo psychological simplification; such simplification is measurable in terms of reduced numbers of errors and latency of response. He described several diverse studies of aesthetics in which repeated exposure to the stimulus or training in the domain leads to altered preference patterns. Walker attributed these changes in preference to the simplification of the event which the subject had experienced. One can speculate that the described simplifications in psychological events were paralleled by changes in problem representations. Under this interpretation, it is clear that problem representation and preference are closely connected.

A Common Scenario as Illustration

In the experience of this author, another common event provides vivid anecdotal illustration of the relationship between preference and problem representation. A typical job interview for a computer science senior with no real-world experience consists, in part, of a set of problems. The problems are similar to the types of problems that the interviewee would solve on the job. The

interviewer describes the problems and asks the candidate to explain how he or she would solve the problem. The candidate may ask additional questions; he or she describes the solution, as well as something about the transformation from the problem to the solution. In other words, the candidate verbalizes some pieces of the problem representation and the process of refinement. Presumably, the candidate's responses determine in part whether a job offer will be made.

On the other hand, after the problem solving exercise, the candidate often knows whether he or she would really like the job. This revelation is often sudden and unexpected. Particularly, if the conclusion is negative, the candidate may have a difficult time pretending to be interested during the remainder of the interview.

Presumably during the problem solving session, some preference judgements were made in reference to the described problems and the process of generating a solution. Because the preference judgements were made unconsciously, the approval or aversion to the job seems sudden and uncalculated.

Summary - Problem Representations and Preference

Two theoretical models have been described. An adequate problem representation and its formation have been suggested as critical elements in the superior problem solving by experts. The formation of a representation includes the selection of an initial representation and an extended refinement process.

Information processing in computer science situations is similar to functioning in the physical environment. One characteristic of environmental functioning is the generation of preference judgements based on informational characteristics of a setting. Informational characteristics of computer science situations may influence preference in a similar way.

Formation of a problem representation and preference appear to be interrelated processes. Situations which are highly coherent enhance the formation of representation; situations with promise help to sustain this same

process.

In the next chapter, the experimental design for the current study is discussed. This design incorporates the theoretical issues which were discussed in this chapter.

Notes to Chapter II

[1] Interaction of content and preference - The model of preference which has been postulated revolves around characteristics of informational organization. Of course, content also affects preference. Unfortunately, content can not be cleanly separated from informational organization; a prefered content might also be characterized by prefered styles of informational organization (cf. Kaplan and Kaplan, 1982).

CHAPTER III

DESCRIPTION OF METHODOLOGY AND TEST INSTRUMENT

The design of an appropriate study has been one of the most challenging issues faced by any human-computer interface researcher. The task is particularly difficult for a person who is working in the area of preference since the body of literature is so small. Because almost no precedents exist, a researcher in this area is forced to start almost "from scratch."

This chapter describes the methodology, test instrument, and procedure which were chosen for this study. In particular, the specific aspects of and rationales for five facets of the experimental design are highlighted and include:

Chosen experimental methodology

Chosen experimental unit

Chosen content of the test instrument

Data collection procedure

Specific description of test instrument

Description of Chosen Methodology

Within the domain of aesthetics in computer science, two methodologies have been used. Molzberger (1983) conducted interviews with expert programmers. Soloway and Ehrlich (1984) collected performance measures from experts and novices. Unfortunately, both of these approaches have serious drawbacks. The interview methodology is difficult to quantify; the results depend entirely on the interpretation of the interviewer. The performance measurement

approach, on the other hand, yields quantitative data. This approach, however, is a highly indirect measure of preference. (It relies on the assumption that performance and preference are parallel operations and that the manifestation of one of the operations is descriptive of the other operation.)

Ratings by subjects have been used successfully in computer science to measure user-perceived quality and satisfaction with computer systems and software packages. (cf. Dzida, Herda, Itzfeldt, 1978; Gilfoil 1982; Rushinek and Rushinek, 1986) This technique has the advantages of being both quantitative and direct, since the subject provides a numerical rating of his/her reaction to an item.

In this study, subjects were asked to provide both preference and interest ratings of the stimulus items. Preference ratings were collected for all of the items and interest ratings were gathered for a subset of the items.

Selection of the Experimental Unit

The size of software systems ranges from the small to the enormous, with some production systems exceeding a million lines of programming code. Given this large range, a second challenge to the experimenter is to select an experimental unit of an appropriate size.

The short, functional program has been used successfully to study several issues in computer science expertise. Simple programs contain many of the basic elements of programming, such as looping, manipulation of variables, input and output, and testing. This experimental unit preserves many of the features of complex software tasks; yet the short programs can be written in a familiar language and format, and held to a manageable size, with meaning for novices, intermediates, and experts.

The short program is especially useful in studying the role of abstract information for experts and novices. In particular, because even simple problems can be correctly solved by many different programs which may have enormous differences in style, statement type, or amount of noise, these items are useful in studies that require more than one solution to a single problem.

Behaviors associated with these functionally equivalent but syntactically different programs have been used to illustrate the facility of experts to "see" beyond syntactic differences. (cf. Kahney, 1983; Soloway, Ehrlich, Bonar, 1982)

Unfortunately, the results of research using small programs may have only limited generalizability. A robust phenomenon for fifty line programs may have little relevance for a million line software system. On the other hand, the use of extremely large software systems as experimental units is potentially both expensive and difficult to control (Sheil, 1981).

This study uses an intermediate form of stimulus, which lies somewhere between the small, tightly controlled program and a large software system. The items for this study consist of problem statements and incomplete program solutions. The problem statements provide a general context for the solutions, with only the information for the given solution specified in detail. The general problems are of sufficient complexity that one could imagine developing software systems which range in size from small to intermediate.

The partial solutions, on the other hand, provide windows into the solutions to the overall problems. These windows are clearly managable since the given solutions consist of between five and twenty-three lines of code. Yet because the given solutions are partial, the larger solutions are not restricted in size as an artifact of the experiment.

Selection of Content of the Test Instrument

The selection of the content for the task actually involved selecting the programming problems and the corresponding solutions. One is challenged to find problems and solutions which are diverse enough to be representative of a broad range of computer science tasks but still reasonably small in number to form a viable experimental task.

Traditionally, much of computer science education emphasizes exposure to a variety of software and hardware problems, algorithms, and data structures. Intuitively, one might suspect that important preference differences for computer scientists are simply related to the type of problem, algorithm, or data structure

which they are working with. The following scenario illustrates one way that the choice of algorithm might be related to preference.

> Programmer X has encountered a situation which calls for a sorting procedure. Given a choice, would she prefer to use a bubble sort or an insertion sort, all other variables in her decision being equal in either case.

In order to cover a broad range of problem and solution types, five programming problem types and ten programming solution types were included in the test instrument. For four out of the five problem types, the solutions were functionally equivalent, but not identical. In the fifth problem type, the problems and solutions were similar.

The problems and solutions for this study were all developed by the experimenter. However, many of the problems and solutions resemble typical textbook examples since textbook items presumably are somewhere near the mainstream of practice.

Choice of Predictor Variables

Two kinds of predictor variables were systematically varied in the test instrument: (a) Surface structure of the solution and (b) Classicalness of the problem and solution. An additional predictor variable, interest, was obtained through ratings.

Surface Level Solution Manipulations - Soloway and Ehrlich (1984) found performance differences between experts and novices on tasks involving violations of rules of discourse. Specifically, when a rule of discourse was violated, the performance of the expert was degraded to nearly that of the novices. Significant differences were found between expert and novice performance, performance in the violate and no violation condition, and the

interaction of expertise and the violation.

The current study included a variation on Soloway and Ehrlich's approach. Soloway and Ehrlich (1984) found significant effects on five rules of discourse. Only two of those rules were tested in the current study, including the Double Duty Rule and the IF-WHILE rule.

Double Duty Rule - The double duty rule states "Don't do double duty with code in a non-obvious way." Many interpretations of this rule are possible, including

a. Any variable which serves two functions is considered to be in violation of this rule. For example, a variable which functions both as part of a Boolean condition for loop control and as an array size is in violation of the rule.

b. A data structure which is used predominantly in one way, but a small component is used in another. For example, in a table data structure, the slots of the table are used to hold data. Sometimes the table size is stored in the first slot in the table.

c. Two variables with nearly identical names, serving in the same capacity is an example of non-obvious double use of code.

Construct Affordances Rule - Soloway and Ehrlich also considered a rule which stated, "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 repeatedly executed." In other words, this rule is violated when an inappropriate construct is used in the described situation.

Several experts have suggested that the IF-WHILE rule is a special case of a more general rule that might be called a "construct affordances" rule (private conversation - see Note 1). That is, a programming construct may "afford" special features that make it particularly appropriate in some situations. Conversely, a "violation" of this rule occurs when a construct is used which is inappropriate in the setting but does not make the program incorrect. Some common Pascal constructs, their normal usage situations, and alternative, "violation" constructs

are described in Table 3.1.

The notion that certain program constructs are appropriate to the setting is not without precedent in the human-computer interface literature. In their "Bug Catelogues", Spohrer, et al. (1985) included a large number of "buggy" novice programs. Several programs were included because the authors felt that the novices had used an inappropriate construct. For example, Tax Problem Bug #6 was included because it include a subroutine parameter that referred to the actual program variable. In this case, reference to a copy of that program variable would have been sufficient for the task since no change was made to the variable.

Classicalness - The notion of classicalness is widely used in domains such as music, art, and history. For instance, even a music novice would likely identify The Kanon in D (Pachelbel) as high in classicalness and "We're Strong for Toledo" as low in classicalness. One way to assign some type of scaling to the problem and solution type variables is to consider how "classical" is the problem or solution type. Classicalness refers to its role in computer science. A problem or solution which is seminal in computer science clearly has high classicalness; one which is known but not a milestone has lower classicalness. In the test instrument, the problem categories vary in classicalness; two of the solutions use the classical algorithm, data structures, and program structure; the other two use an alternative algorithm, data structure, program structure, or some combination.

In an attempt to capture this "classicalness" dimension, each problem and solution has an assigned a classicalness rating. (see Note 2 for a description of the classicalness ratings for the test instrument items) The rating scale is:

H (high): found in many introductory programming and data structure texts as a classical problem or solution.

M (medium): found often in introductory programming or data structure texts as a typical example but not as a classical problem or solution.

L (low): found in an introductory programming or data structure text as an

CONSTRUCT	NORMAL USAGE	"VIOLATION"
FOR	when number of repititions is known in advance; automatically maintains a loop counter	WHILE requires that programmer explicitly maintain loop counter and test exit condition.
IF	Boolean condition, followed by statement body executes exactly once	WHILE requires that within the statement body, the exit condition changes; infinite loop otherwise
REPEAT	Statement body executes an unknown number of times, but at least once.	WHILE requires an unnecessary test before the first iteration.
WHILE	Generalized loop; appropriate when number of iterations is unknown.	REPEAT assumes at least one iteration of loop
FUNCTION	A single calculated value is returned from a called subprogram.	PROCEDURE requires the value to be explicitly returned as a variable parameter.

Table 3.1 - Constructs Affordances and Violations

exercise or an unusual solution.

Interest - In addition, a third independent variable involved the question of how interesting the items appeared to be. Other studies of preference, outside of computer science, have found that interest seemed to be a useful predictor of preference (cf. Crozier, 1977; Hare, 1974).

Description of the Data Collection Procedure

Five types of programming problems were included in the test instrument (matrix rotation, sorting, main procedure, numerical calculation, and searching). The problems were stated in the context of a larger software engineering situation. For each problem, a partial Pascal solution was included. The solutions were syntactically correct.

The test instrument consisted of twenty problem and solution pairs. For each pair, the subject was instructed to "rate the program segment" on a five-point scale. The points on the scale included "unsatisfactory", "marginally acceptable", "adequate", "highly acceptable", and "elegant."

Following the twenty problem and solution pairs, the subjects were asked to complete eleven demographic questions. These questions were followed by a repetition of ten of the problem and solution pairs from the first part of the instrument. For these ten problems and solutions, the subjects were asked to "rate how interesting you found this problem/solution pair on a scale from 'A' to 'E'," where "A" was labeled as "highly uninteresting" and "E" was labeled as "extremely interesting". Appendix B contains a facsimile of the test instrument.

The items in the test instrument were counterbalanced. Two different orders of preference items were used and four different orders of interest items, making a total of eight different orders of presentation.

The data collection proceeded as follows: The experimenter was introduced (truthfully) to the subjects as a faculty member in the computer science department. The subjects received a short, written description of the current project (a copy of this description is included in Appendix B). This description

was reviewed by the experimenter. Consent forms were distributed and explained. The subjects who wished to participate signed and dated the forms.

The subjects then received a test booklet which included a directions page, as well as the preference, demographics, and interest items. The directions page contained an explanation of the way in which the subjects were to complete the task, as well as a statement which informed the students that they would have one hour to complete the task. The experimenter reviewed the directions page and asked for questions. The subjects then completed the task and returned both their test booklets and consent forms to the experimenter. At this time, they left the room.

Description of the Sample

The task was administered to 47 students in a sophomore computer science class at Bowling Green State University (BGSU) and to 46 students in a senior level computer science class at BGSU. The task was administered at the first scheduled meeting of the respective classes. The students were given 1 hour to complete the task, although none of the participants required all of the allotted time. The students were not paid for their participation; however, they were given the option to leave without participation in the study. In general, the novices in the study were the students in the sophomore class and the experts were the students in the senior class. However, because students occasionally take these courses out of order, the expertise level of the students was defined by the number of computer science courses that the student had completed. These novices were students who had completed 3 or 4 computer science courses (these courses most likely included Introduction to Programming, Advanced Programming Techniques, Assembler Language Programming, and possibly COBOL programming or programming languages). The experts had completed 6 or more computer science courses. Since most of the students in this category were computer science seniors or graduate students, they had presumably completed more than 6 computer science courses (see: Note 3 for further discussion of measurement of expertise in this study. Note 4 for description of

pretesting procedures.). Appendix C includes a detailed description of the sample characteristics.

Description of the Problems and Solutions

Matrix Rotation Problems - Matrices are widely used in mathematical problems. The implementation of matrices and their operations are common programming problems. The operations which are usually associated with matrices are addition, transpose, inversion, and multiplication.

For the matrix problems that were used in this study, a more unusual operation was selected. The problem was to write a program that rotates a matrix ninety-degrees. The rotation of a matrix is a useful operation in problems in which the matrix represents some kind of physical object which needs to be turned. Because the matrix rotation is an unusual operation, the problem is considered to be of low classicalness.

Matrices are implemented in several different ways. Because a matrix is a two-dimensional structure, the most common implementation is with a two-dimensional array. Two of the matrix rotate solutions in the test instrument use this implementation.

The implementation of the matrix with a two-dimensional array is highly classical. Although the problem in this case has low classicalness, the implementation of a matrix with a two-dimensional array in general is the usual approach.

A less common implementation is with a one dimensional array representing the matrix. In the one-dimensional array approach, the matrix is stored by rows or columns. That is, all of the elements of row (column) one are in their normal matrix order, followed by all of the elements of row (column) two, and so forth. The one-dimensional array implementation of matrices is actually used in practice in some programming languages, most notably FORTRAN. Two of the matrix rotate solutions in the test instrument use this implementation.

The implementation of a matrix with a one-dimensional array is unusual; the solutions of this type are rated as low in classicalness. The complete

description of the matrix rotate problem and solutions are shown in Table 3.2.

Sorting Problems - One of the most common problems in software development is to sort a list; sorting algorithms are typically a focus of a data structures course. The body of sorting algorithms is large and diverse and ranges from the highly familiar to the obscure. Not surprisingly, this problem type is considered to have a high level of classicalness.

The bubble sort is perhaps the most widely known sorting algorithm, and hence is high in classicalness. In this sort, the program makes two passes over the list; the smallest (largest) items "bubble" to the top in a list which is sorted in ascending (descending) order. This sort is found in nearly every introductory programming text because it is straightforward, although not computationally efficient.

The insertion sort is included less often in introductory programming texts, but is covered often enought to be assigned a medium classicalness rating. In this sort, each item of the list is inserted into its proper position. Like the bubble sort, the insertion sort is not particularly efficient.

The sorting problems and solutions are described in Table 3.3.

Main Procedures - Within computer science, students are taught to write structured programs. In a structured program, the main procedure initiates other parts of the program to solve the problem. Little, if any, of the problem itself is solved in the main procedure.

To generate a main procedure is a fairly classical computer science problem in an abstract sense. However, unlike the sorting or matrix problems, the generic problem statement of "...write a main procedure..." is not meaningful without the larger context of a specific target problem. In the problems in the test instrument, the task of writing a main procedure is cast in the setting of larger software development problems, which were uniquely designed by the experimenter. Because the generic task of generating a main procedure is highly classical, but generating the specific main procedures for the test instrument is less classical, these problems are considered to be at a medium classicalness level.

PROBLEM	SOLUTIONS	VIOLATES DOUBLE DUTY RULE?	VIOLATES CONSTRUCT AFFORDANCE RULE?	PROBLEM CLASS. LEVEL	SOLUTION CLASS. LEVEL
MATRIX ROTATE	1. JUNGLE ESCAPE - board game. The object-to rotate the gameboard 90°.	NO	WHILE replaces FOR	LOW	LOW
	2. SOUTH POLE - navigation of the South Pole where every direction is north. Rotate map of the South Pole.	NO	WHILE replaces FOR	LOW	HIGH
	3. ARRANGE HOUSES Rotate a grid that represents a house, 90°.	YES	NO	LOW	LOW
	4. QUILTING - Rotate quilt frame 90° for quilting	YES	NO	LOW	HIGH

Table 3. 2 - Matrix Rotate Items

PROBLEM	SOLUTIONS	VIOLATES DOUBLE DUTY RULE?	VIOLATES CONSTRUCT AFFORDANCE RULE	PROBLEM CLASS. LEVEL	SOLUTION CLASS. LEVEL
SORT A LIST (ascending order)	1. CONTRIBUTIONS-sort charitable contributions in ascending order	NO	NO	HIGH	HIGH
	2. WEIGHTS - sort dog weights for a dieting dog	NO	NO	HIGH	MEDIUM
	3. PRICE LIST - sort list of prices from a four family garage sale	NO	WHILE replaces IF	HIGH	HIGH
	4. SEAWATER - sort magnesium concentrations in manufactured seawater	YES	REPEAT replaces WHILE	HIGH	MEDIUM

Table 3. 3 - Sorting Items

Each of the solutions of this type in the experiment is interactive and can be executed as many times as the user would like. Since only the main procedure is given, the subjects never see the entire solution to the problems.

The solutions vary in how much of the problem is actually solved in the main procedure. The modular solutions, which act only as "drivers" for the rest of the program, are completely consistent with the structured programming paradigm. These solutions have a high classicalness level. The two solutions which act as "drivers," but also contain a few lines of code which solve part of the problem are less frequently presented in textbooks. These types of main procedures usually appear as examples of poor programming practice. The main procedures with a few details are medium classicalness solutions.

Table 3.4 details of the Main Procedure problems and their associated solutions.

Numerical Calculations - Many software systems include some numerical calculations. These typically can stand alone and are isolated into procedures or functions.

The problems which are included in the experiment call for a numerical calculation; the calculation is either a generation of a number or a tabulation of a sum. Instead of simply stating a numerical problem like "add a series of numbers", the numerical problem is presented as part of a small story. Like the main procedure problems, the calculation problems are set in and inseparable from the larger programming problem in the problem statement. The problem type is considered to be at a medium classicalness level.

Many numerical problems can be solved either recursively or iteratively, although recursion is typically more problematic. Two of the solutions to these problems are recursive; two are iterative.

The recursive numerical calculations are each assigned a low classicalness rating since neither of the underlying algorithms are inherently recursive in nature. By contrast, the iterative subprograms are considered to be highly classical because the underlying algorithms are inherently iterative.

PROBLEM	SOLUTIONS	VIOLATES DOUBLE DUTY RULE?	VIOLATES CONSTRUCT AFFORDANCE RULE	PROBLEM CLASS. LEVEL	SOLUTION CLASS. LEVEL
WRITE A MAIN PROCEDURE	1. LIBRARY SYSTEM-write a library system to help undergrads generate term papers	NO	NO	MEDIUM	MEDIUM
	2. SHOE STORE-write an online order facility for the Technocrat On-line Shoe Company	NO	WHILE replaces REPEAT	MEDIUM	MEDIUM
	3. PET POISON HOTLINE write an answering system for a pet poison information clearinghouse	NO	NO	MEDIUM	HIGH
	4. THUNDER STORM-write a thunder storm locator service for pilots	NO	WHILE replaces REPEAT	MEDIUM	HIGH

Table 3. 4 - Main Procedure Items

The structure of the numerical calculation problems and solutions are shown in Table 3.5.

Searching Problems - Like sorting problems, searching problems are a common class of software problems. Many searching algorithms exist; the topic of searching is central in a course in data structures.

The problems which were used in the test instrument require a search of a table. This is a highly classical problem. The solutions include two types of variations; they differ in the searching algorithm and in the underlying data structure.

The exhaustive search is the most intuitive searching strategy. In an exhaustive search, each item in the table is considered until the target is found or the table is exhausted. The exhaustive search requires no maintenance of the table; however, it is not efficient. Because of its lack of efficiency, the exhaustive search is not emphasized. The exhaustive search solutions are given lower classicalness ratings.

The binary search, in contrast, is more efficient but less intuitive. In a binary search, the whole table is searched, then half the table, then one-quarter of the table, and so on, until the target is found, or no objects are left. The binary search requires that the table be maintained in order. Because of its computational efficiency, the binary search is well-known; it is considered to have high classicalness.

The table data structure can be implemented in at least two ways. The entire table is part of the same data structure in an implementation with records; in this implementation, complex variable names are typically required. If the table is implemented with multiple parallel arrays, each table item corresponds to an array. In either case, the programmer is responsible for maintaining the table during any table update.

Table 3.6 summarizes the details of the searching problems and solutions.

Summary of Methodology, Test Instrument, and Procedure

The collection of preference ratings was selected as an appropriate

PROBLEM	SOLUTIONS	VIOLATES DOUBLE DUTY RULE?	VIOLATES CONSTRUCT AFFORDANCE RULE	PROBLEM CLASS. LEVEL	SOLUTION CLASS. LEVEL
NUMERICAL CALCULATIONS	1. TEST GRADING- tabulate a score on an exam	NO	NO	MEDIUM	HIGH
	2. LEGISLATIVE VOTE- tabulate a legislative vote	YES	NO	MEDIUM	LOW
	3. SECRET CODE- generate a secret code	NO	PROCEDURE replaces FUNCTION	MEDIUM	HIGH
	4. LOTTERY NUMBER- generate a winning lottery number	YES	PROCEDURE replaces FUNCTION	MEDIUM	LOW

Table 3. 5 - Numerical Calculation Items

PROBLEM	SOLUTIONS	VIOLATES DOUBLE DUTY RULE?	VIOLATES CONSTRUCT AFFORDANCE RULE	PROBLEM CLASS. LEVEL	SOLUTION CLASS. LEVEL
	1. JEWELRY STORE- update parts catelogue for a jewelry store inventory	YES	NO	HIGH	LOW
	2. ARCHEOLOGY- maintain an inventory of an archeological dig	YES	REPEAT replaces WHILE	HIGH	HIGH
	3. TOXIC SUB.S- maintain an employee inventory for contact with toxic substances	YES	WHILE replaces IF	HIGH	LOW
	4. RARE WILD FLORA- maintain an inventory of endangered plants	YES	NO	HIGH	HIGH

Table 3.6 - Searching Items

methodology since it produces direct quantitative measurements of the preference phenomenon. The test instrument for the experiment includes several *a priori* predictor variables and samples preference, interest, and a variety of demographic variables.

Notes to Chapter III

[1] The notion of construct affordances was suggested by Frank Cioch. Some examples were suggested in private conversations with R. Belew, A. Gilles, K. Ross, and M. Weaver.

[2] The classicalness levels for the items in the test instrument were originally assigned by the experimenter. Private conversations with R. Belew, A. Gilles, K. Ross, and M. Weaver provided support for the assignments

[3] Other studies of computer science expertise have used freshmen in their first computer science course as novice subjects. The topics and difficulty level of the items in the test instrument precluded the use of introductory students.

[4] The test instrument was pretested with three novices and two experts. The subjects in the pretest were paid $3.50; they required between 30 and 60 minutes to complete the task. While the subjects in the pretest found the task generally interesting, they found several syntax errors in the test instrument. These errors were subsequently corrected.

CHAPTER IV

PREDICTION OF AESTHETICS IN EXPERTS AND NOVICES

The results of this study are presented in this chapter and the next two which follow. The current chapter highlights the roles of the independent variables (interest, discourse rule violation status, problem and solution classicalness) in predicting the preferences of the experts and novices. The two chapters which follow emphasize the patterns of preferences of and their relations to the perceptions of the two subject groups.

Analyses

Across all 20 of the items in the test instrument, experts had higher preference ratings than the novices (means of 3.27 and 3.14, respectively). This difference is significant ($t=3.20$ and $p<.005$). At the same time, however the mean ratings for the two groups were highly correlated ($r=.82$). This high correlation indicates that the two groups found the same problems highly preferable or less preferable. One is tempted to conclude that a higher expertise level leads one to greater appreciation (as reflected by higher preference) of the items, but that the general pattern of preference is similar between the two groups.

With respect to interest ratings, a different picture emerges. Although the interest ratings are virtually identical across the two groups (average rating around 2.9 suggests that neither group found the items highly interesting), the correlation between the two sets of ratings is -.04. In other words, what constitutes interestingness is completely different between the two groups.

These results make it clear that the roles of the interest and the other

independent variables (problem and solution classicalness, discourse rule violation status) must be examined separately for the novices and experts. The type of analysis which was used to predict the preference ratings was analysis of covariance. These analyses, performed for novices and experts, used mean preference ratings for each item as the dependent variable. Mean interest ratings for each item were also computed separately for each group. The other three independent variables (problem and solution classicalness and discourse rule violation status) were the same in each analysis, since they involved characteristics of the items rather than of the subjects.

In the analysis of covariance models, interest, as an interval variable was the covariate. The other three independent variables were used as categorical variables. Discourse rule violation status was coded as 0 if the item had no violations, and as 1 if the item had one or more violations. Solution classicalness was assigned a value of 0 if the item was at a low level of classicalness, and had a value of 1 if the item was at a medium or high level of classicalness. The value of problem classicalness was 0 if the level was low or medium, and was 1 if the level is high. (See Note 1 for a more detailed description of the analyses which were performed. Appendix E contains a facsimile of the data which was used in the analysis.)

Before presenting the results from the analyses, it is useful to consider a general framework which organizes the four independent variables. Classicalness, in this study, was divided into both problem and solution aspects. This division is also a useful way to consider the other two independent variables. The discourse rule violation status is oriented to solution issues. Interest, however, is more closely related to the problem than to the solution. Thus, as Figure 4.1 illustrates, the four variables can be considered as cells in a 2 x 2 matrix. The matrix includes both the solution vs. problem dimension and the classicalness vs. the other more surface level issues.

Interest was included as an aspect of problem on the basis of a principal components factor analysis of the four independent variables, performed separately for the experts and novices. For both groups, two orthogonal factors

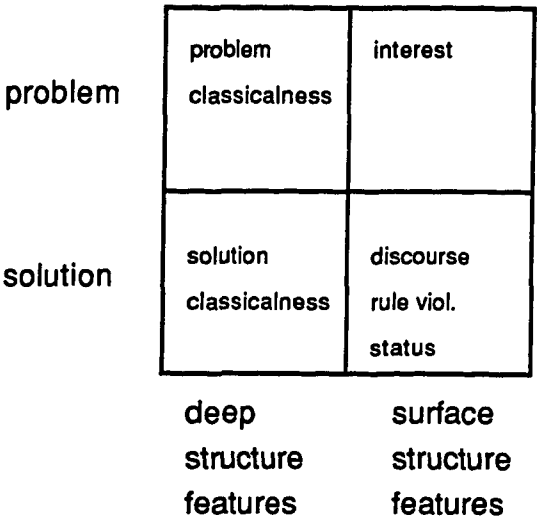


Figure 4.1 - Prediction Framework
for independent variables

emerged, using a minimum loading criterion of .40. Solution classicalness and discourse rule status had high loadings on Factor 1 (.84 and -.82, respectively for the novices and -.82 and .84 respectively for the experts). Interest and problem classicalness were strongly loaded on Factor 2 (.81 aand -.70 respectively for the novices and -.74 and .77 respectively for the experts). Appendix D contains correlation matrices, loadings from rotated factor matrices, and plots of loadings vs. variables for both groups.

Results

The amount of variance accounted for by the model for each subject group was roughly comparable; R-squared = .64 and .68 for experts and novices, respectively. For both groups, the overall F ratio for the model was significant ($F(4,15) = 6.70$ $p < .003$ for the experts. $F(4,15) = 7.95$ $p < .002$ for the novices.) (Appendix F contains the details of the analysis.)

Using the prediction framework which was presented in Figure 4.1, Figure 4.2 shows the partial correlations between each independent variable and preference for the two subject groups. As these correlations suggest, interest and solution classicalness are strong positive predictors in both cases, although solution classicalness seems to have become slightly less important for the more expert subjects. The roles of the other two independent variables differ in the two cases. Discourse rule violations are important points of focus for novices, but become less important with increasing experience. By contrast, problem classicalness, becomes increasingly more salient with greater experience.

In the context of the prediction framework, increased expertise is marked by movements in emphasis. As expertise increases, the trend in focus tends to be more on problem characteristics than on solution attributes. Additionally, as expertise increases, focus moves away from surface level features, in the direction of deep structure characteristics. This result is consistent with other non-preference studies of expertise in computer science. Such studies have repeatedly shown that increasing expertise is accompanied by processing of

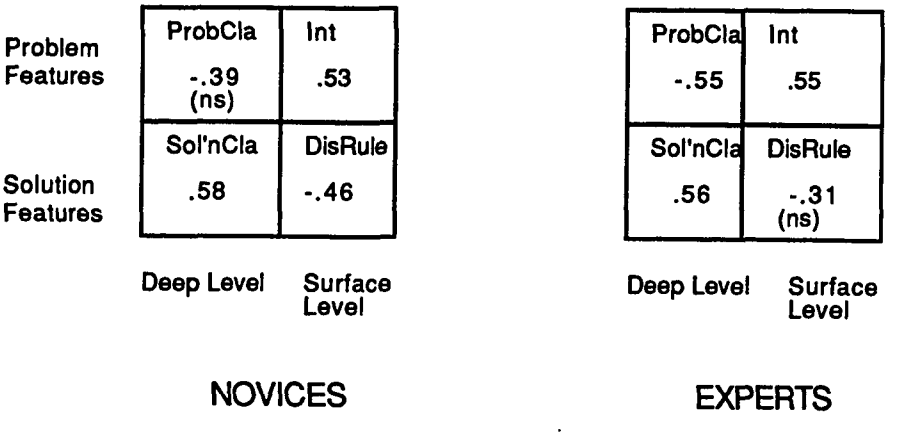


Figure 4.2 - Partial correlations for Experts and Novices using prediction framework from Figure 4.1

increasingly abstract information.

Preference and Problem Representations

In Chapter II, it was suggested that experts and novices differ strongly in their patterns of formation and use of problem representations. In this section, the differences in the roles of the independent variables for the two groups is discussed in the context of problem representations.

Expert Problem Representations

In Chapter II it was posited that experts invest much of their problem solving activity on the formation of an adequate problem representation. This process appears to involve two steps. First is the formation of an initial representation, using well-developed perceptual skills. This is followed by a refinement of the representation. Again, the expert relies on perceptual skills; the expert continues to depend on useful environmental patterns while refining the representation. Numerous studies have indicated that the generation of a solution by an expert follows straightforwardly from the formation of the representation.

The preference ratings of the experts in the current study were based in large part on problem characteristics. These results seem to be predictable and follow directly from the proposed theoretical model from Chapter II. Previous studies have suggested that experts focus on formation of problem representation during problem solving. The current results suggest that experts focus primarily on the problem, even when they are already presented with a solution. Presumably the expert forms a problem representation on the basis of problem elements. While solution characteristics apparently enter into the problem representation to a lesser degree, they do not seem to be central features of the representation. The solution information that is included is abstract. In this study, solution classsicalness, which was an abstract feature of the items, was important to the experts. The surface level solution features were apparently treated as noise and ignored.

These results suggest that experts do make preference judgements while they are forming their problem representations. In this respect, preference judgements resemble the other problem solving activites of experts which are similarly involved in problem representations.

Novice Problem Representations

Unlike experts, novices do not focus on the formation of a rich problem representation. Instead they appear to focus more on the solution. The solutions of novices, as a consequence, without a rich representation as an antecedent, tend to be more limited.

The results of the current study support this notion. Apparently novices not only begin to solve a problem by concentrating on the solution, but they also make preference judgements based on solution characteristics.

Expert and Novice Problem Representations

In this study, experts and novices focused on different elements of the stimuli. For the experts, problem characteristics played a central role in preference judgements; for novices, solution features were more pivotal. Furthermore the expert preference judgements seem to have been strongly influenced by deep features in the items, as indicated by the importance of problem and solution classicalness as predictor variables. By contrast, the preference ratings of the novices appear to have been strongly influenced by surface level features of the items (specifically, interest and discourse rule violation status). These results appear to be related to differences in problem representations as a function of expertise.

Summary

Two analyses of covariance of expert and novice preference ratings were performed. Preference for the experts was based primarily on problem characteristics; experts also seem to focus on abstract elements in the stimulus. For the novices, solution features were critical predictors of preference. Novices

tend to focus on surface level features of the stimulus as well. One interpretation of these results is that the problem representations of experts are of a different nature than those of novices in terms of both sophistication and type of components.

Notes to Chapter IV

[1] Analysis of Covariance - One kind of analysis which is appropriate for a study which includes both nominal and interval data is called an analysis of covariance. Such an analysis incorporates features of both analysis of variance (appropriate for nominal data) and regression analysis (appropriate for interval data). In an experiment in which the independent variables involve both types of data, as the current study did, this analysis technique permits all of the independent variables to be considered together. (cf. Blalock, 1979; Wildt and Ahtola, 1977).

The current analyses were performed using a general linear model procedure in the commercial statistical package, SAS. Variables of discourse rule violation status, solution classicalness, and problem classicalness were encoded as dummy variables.

CHAPTER V

PERCEPTUAL CATEGORIES: EXPERTS

The preceeding chapter described differences between experts and novices based on the magnitudes of their preference ratings. In particular, analyses for each group of subjects revealed that problem features and solution features strongly influenced the ratings of the experts and novices, respectively. These results suggested that the problem representations for the two groups were different. In this chapter and the next, the focus is not on amounts of preference, but rather on patterns of preference for the two groups.

What can be gained from understanding about patterns of preference? The answer to this question has two parts:

1. Peoples' preferences are an expression in some sense of the way in which they perceive. As the discussion in Chapter II indicated, perception seems to change with increasing expertise. By considering the collected preference ratings in the context of perception, it is possible to better understand the role of expertise in this study. That the problem representations of experts and novices differ is an indication that their perceptions, which are essential for the development of problem representations, are likely to differ as well.

2. A vital aspect of perception is that entities which are perceived are perceived as members of categories. Clearly perception would not be effective if each encountered object was experienced as a "one-of-a-kind." The mechanisms of perception take advantage of various similarities in inputs, so that items are experienced as being one of a group.

Considering patterns of preferences of experts and novices allows one to begin to understand the nature of the underlying perceptual categories of the two groups.

There are statistical procedures which extract patterns from a group of responses. The particular procedure which was used in this chapter and the next is called the Guttman-Lingoes Smallest Space Analysis (SSA-III). Using SSA-III, the preference ratings of the 20 problem and solution pairs were grouped. Items were considered to be elements of a factor if they had a loading greater than plus or minus .38. Items which were loaded onto more than one factor were not included in either factor. (See Note 1 for a discussion of factor analysis in general and SSA-III in particular.)

As the results in Chapter IV indicated, different aspects of the test instrument items influenced the responses of experts and novices. In particular, experts focused on the problem characteristics and novices highlighted the solution characteristics. (Variables of interest, problem classicalness, and solution classicalness influenced expert preference and variables of interest, solution classicalness, and discourse rule violation status influenced novice preference.) In this chapter, the factors which describe patterns of expert preference will, not surprisingly, be interpreted primarily in terms of problem characteristics. In the next chapter, patterns of novice preference will be interpreted principally in terms of solution characteristics.

Expert Factor Structure

Using the expert preference ratings, the SSA procedure yielded four factors. Fourteen of the 20 items were included in the factors. The four categories were: the UNIQUE PROBLEM Category, the NOVEL USE OF A COMMON CONSTRUCT Category, the OFF OF THE SHELF Category, and the INTRODUCTORY PROGRAMMING Category. Table 5.1 provides the names of the items within each category, their loadings, and information on their levels of the variables of interest, problem classicalness, solution classicalness, and discourse rule violation status. The next four sections describe each of the categories in detail. Appendix G contains the details of the factor analysis.

FACTORS	LOADINGS	PROBLEM CLASS.	SOLUTION CLASS.	MEAN INTEREST	DISC. RULE
1. UNIQUE		33% high	100% not low	3.15	66% violation
Pet Poison	-.7587	not high	not low	3.00	no viol.
Thunder Storm	-.4922	not high	not low	3.24	violation
Archeology	.7206	high	not low	3.22	violation
2. NOVEL USE		0% high	75% not low	3.00	75%violation
Quilting	.5581	not high	not low	2.81	violation
Test Grade.	.4224	not high	not low	2.79	no viol.
Jungle Esc.	.7351	not high	low	3.25	violation
South Pole	.3884	not high	not low	3.15	violation
3. OFF SHELF		100% high	66% not low	2.79	66%violation
Price List	-.6326	high	not low	2.48	violation
Toxic Sub.s	-.4167	high	low	3.14	violation
Contribu.s	-.7953	high	not low	2.76	no viol.
4. INTRO.PROG		50% high	50% not low	2.82	50%violation
Lottery No.	-.4311	not high	low	3.00	violation
Weights	-.3998	high	not low	2.64	no viol.

Table 5.1

EXPERT FACTOR CHARACTERISTICS

UNIQUE Category

The UNIQUE category contains 3 problems which have not been solved before. That is, neither the problem statements nor the solutions are likely to be commonly found in textbooks or software catalogues. Two of the three problems statements request interactive software systems with real world applications; the problems are to develop a thunder storm locator system for pilots and a pet poison hotline management system. The third problem is also set in a real world setting; it is to design a software system for an archeologist who is conducting a local dig.

In each of these problems, the task problem asks for a specific, small part of the entire software system. For the interactive programs, a main procedure is needed; for the archeology problem, a search routine is required. The problem and solution classicalness levels for the three items are medium or high, and the interest levels are moderate or high.

It is important to notice that the factor loadings for the items in this category, shown in Table 5.1, are both positive and negative. The signs for the main procedure problems loadings are negative. The sign for the archeology problem loading is positive and in the opposite direction from the loadings of the other two items. This means that the archeology problem anchors one end of a dimension, while the other two problems are at the opposite pole. Since all three items involve the larger context of a real-world problem that has not been solved before, rather than the specific small task problem and its characteristics, the common theme seems to be related to uniqueness .

NOVEL USE OF A COMMON CONSTRUCT Category

The four problems in this category call for an unusual use of a familiar construct. In three of the problems, the construct is a matrix and the problem is to rotate the matrix 90°. This rotation is uncommon since it has few applications outside of the manipulation of spatial objects. The fourth problem involves the calculation of grades in a subprocedure. One can guess that to students, the calculation of grades is always an unusual process, in any form.

The items in the category have medium or low problem classicalness and moderate or high interest levels. The levels of solution classicalness vary from low to high. However, the unifying feature of this group of items seems to be that a novel problem is solved in a conventional way.

OFF OF THE SHELF Category

Many problems in computer science arise so frequently that solutions are readily available in both textbooks and from software distributors. A programmer who encounters such a problem typically purchases a completed solution or codes the solution directly out of a textbook. Searches and sorts are problems of this type.

The problems in this category include two sorting problems and one search problem. The problems have high classicalness levels and low interest levels. The solution classicalness levels range from low to high. However, the items in this category seem to be united by the prominence of the searching or sorting problem. This appears to be true, in spite of the fact that the problem statement includes a larger context for the problems. None of the included solutions are computationally efficient, but they all involve algorithms which are intuitive and widely-known.

INTRODUCTORY PROGRAMMING PROBLEMS Category

In spite of the efforts of authors of introductory programming textbooks and programming instructors to be original and creative, there has emerged within computer science a style of introductory programming problems. These are the problems which are used as classroom examples or programming assignments. Their sole purpose is pedagogic; their solutions are never intended for use.

These problems typically are in the context of a story or a larger problem but their sole purpose is to illustrate a programming concept. Consider the following example of such a problem.

Given a list of English course ID's and class sizes, find the smallest class size, largest class

size and average class size. (Dale and Orshalick,1983)

The purpose of this problem is for students to practice data manipulations and arithmetic computations, rather than to develop software which might potentially be used.

The fourth expert category contains one subprogram problem and one sorting problem. The subprogram problem is a generator for a lottery number and the sorting problem is a sorter for daily dog weights. The problem classicalness levels are medium and high. Interest appears to be generally low and solution classicalness levels are low and medium.

One common theme in these problems appears to be their similarity to introductory programming problems. Although each of the problems is presented as part of a larger programming problem, the problems are contrived and unrealistic.

Relation of Categories to Preference

The means for preference are compared between categories in Table 5.2. As Table 5.2 illustrates, the UNIQUE category has the highest mean preference score, followed by the NOVEL USE OF A COMMON CONSTUCT category, the OFF OF THE SHELF category, and the INTRODUCTORY PROGRAMMING category. The differences among the category means are highly significant ($F(3,162) = 5.14$, $p < .002$).

Expert Categories and Environmental Preference Framework

In Chapter II, a framework for environmental preference was presented. In this framework, preference is accounted for by two components: engagement and making sense. Each of these components is meaningful in both an immediate and a future context. In this section, the results of this study for the expert subject group are compared to the environmental preference framework.

CATEGORY	# OF ITEMS	PREFERENCE
UNIQUE	3	3.49
NOVEL USE	4	3.44
OFF SHELF	3	3.25
INTRO. PROG.	2	3.03

Table 5.2

Mean Preference Levels for
Expert Categories

Engagement and Making Sense for Computer Science Experts

The categories of the expert factor structure conceptually are distinguishable along at least three dimensions: the scope and "real worldness" of the problem, the novelty of the problem, and the novelty of the tools which will be needed to solve the problem. The items in the UNIQUE category are real world problems with a large scope; both the problems and the required tools are new, since the problems are unique. The items in the NOVEL USE OF A COMMON CONSTRUCT and the OFF OF THE SHELF categories are still real world problems, but with a greatly reduced scope. In the NOVEL USE OF A COMMON CONSTRUCT category, the problems are new but the tools are familiar. In the OFF OF THE SHELF category, both the problems and the tools are familiar. The items in the INTRODUCTORY PROGRAMMING category are neither real-world nor large in scope. The problems are familiar and require the use of familiar tools. (see Note 2)

The level of preference judgements parallels the levels of these issues of scope, novelty, and real-worldness. For real-world problems which are new and of large scope, the preference ratings are high. As the scope, real-worldness, and novelty decrease, the preference ratings also drop. Similarly groups of items which are new and of large scope (UNIQUE PROBLEM category) had higher mean preference than those which contained items of more limited scope (OFF OF THE SHELF category). One can infer that the experts in this study preferred problems which they recognized as having actual use and which challenged them to explore new alternatives. Moreover, problems which resembled each other in terms of these characteristics were perceived as being similar.

These issues, scope, novelty of problem, and novelty of tools, seemed to be reflected in the variables which effectively predicted expert preference. The items which appear to be novel and have extensive scope (members of the UNIQUE PROBLEM category) also have moderately high interest levels and lower levels of problem classicalness. Similarly items which employ novel tools (elements in the NOVEL USE OF A COMMON CONSTRUCT categories) have higher interest levels and lower problem classicalness. The items which have

limited scope and novelty (members of OFF OF THE SHELF and INTRODUCTORY PROGRAMMING categories) have lower interest ratings and tend to include highly classical problems.

By reconsidering the framework for environmental preference which was presented in Chapter II, one is struck by the similarity between the issues of scope and novelty from this study and engagement in that framework. Within that framework, an engaging environmental setting is one which offers opportunities to learn. Within the computer science setting, sampled in this study, problems which are novel and of broad scope would seem to similarly offer possibilities.

The three categories of UNIQUE PROBLEM, NOVEL USE OF A COMMON CONSTRUCT, and OFF OF THE SHELF, also seem distinguishable from the final category of INTRODUCTORY PROGRAMMING PROBLEMS in another sense. The two items in this latter group, unlike the items in any of the other categories, include solutions which are inappropriate for the problem. For example, in the Lottery Number item, the problem was to generate a numerical value (the medium problem classicalness value is an indication that this is about an average, "run-of-the-mill" computer science problem). The solution however, given the averageness of the problem, is unexpectedly obscure, since it includes unnecessary and inappropriate recursion. In the same sense, the other element of this category contains an inappropriately obscure solution, given the familiarity of the problem. The solutions to the problems in the other categories seem to fit the problems more appropriately. (Even though several of the categories include items with unusual solutions, these are solutions to unusual problems).

From the previous chapter, it is clear that for the expert subject group solutions of low classicalness tend to have lower preference. That the categories are not distinguishable in terms of solution classicalness levels alone suggest that low solution classicalness is not the issue. Rather for this group of subjects, the fit between solution and problem emerges as important. Items with inappropriate solutions for the stated problems seem to be perceived as similar. The environmental preference framework which was described in Chapter II also includes a component of fit, called in that terminology "making sense." It seems

reasonable that the fit issue which, in part, distinguishes the INTRODUCTORY PROGRAMMING PROBLEMS category from the other categories is one manifestation of making sense in the computer science arena.

It is useful to notice that in addition to similarities between patterns of environmental preference and the patterns of preference in this study the direction of preferences are similar. Higher levels of making sense and engagement leads to higher levels of preference in both cases.

Preference in a Present and Future Sense

In the environmental preference framework both making sense and engagement have distinct manifestations for both present and future. The indicators for the present describe surface level characteristics of scenes, and for the future describe deep level features. Is that pattern repeated in the responses of the expert subjects in this study?

To answer this question, it is useful to reconsider the role and process of formation of problem representation for experts. Recall that experts concentrate much of their problem solving activity on the formation and refinement of an effective problem representation. Initially this process involves the extraction of salient environmental stimuli. This is followed by repeated references to the stimuli as the representation is refined. Of course both processes are highly dependent on perception (which in this study was captured to some extent by preference ratings). During the formation of the initial representation, perception of surface level features occurs. As the representation is refined, embedded information is extracted from the environment.

In this study, the variables which affected expert problem representations differ in their apparent embeddedness into the items. (see Figure 4.1) Because the experts in this study were generally computer science seniors and the items were at a difficulty level appropriate for computer science sophomores, one can guess that the experts in this study were able to form initial problem representations. According to the analysis in Chapter IV, the one measured variable which seemed to be involved in the initial representation of the experts

was the variable of interest. Interest is a surface level variable because it seems to reflect how well the items "caught the eye" of the subjects. Because interest also seems to be tied to the degree of engagement for an item, a reasonable inference is that interest in this study was a surface level component of engagement.

No variable measured in this study seemed to distinguish the items on the basis of fit in an immediate sense for the expert subject group. However, because of the difficulty level of the items for this subject group and the apparent ability of the subjects to form initial representations, it seems reasonable to suppose that all of the items made sense in a surface level context.

The two classicalness variables initially were presented as being embedded features of the items. As the discussion in this chapter suggests, for the experts, the key features of items at an embedded level are problem scope and novelty (indicated by problem classicalness) and solution appropriateness (as indicated by the match between solution classicalness and problem classicalness). A possible scenario is that the expert, while forming and refining an adequate problem representation over time, evaluated both the scope of the problem and the appropriateness of the solution for the problem. In order to make these types of evaluations, the expert would have to go "deep" into the presented item. In other words, it seems unlikely that the experts determined the appropriateness of a solution or problem scope immediately upon seeing the problem and solution pair.

In the previous section, it was suggested that problem scope and novelty, and solution appropriateness are facets of engagement and making sense, respectively. In addition, the items seem to reflect deep level features of these components.

All of the features of the environmental preference model, then, may have appeared in this study for the expert subject group. The matrix in Figure 5.1 describes a proposed model of making sense and engagement, in both a surface and deep level for the items and the experts in this study. Although the components of environmental preference adopted characteristic computer

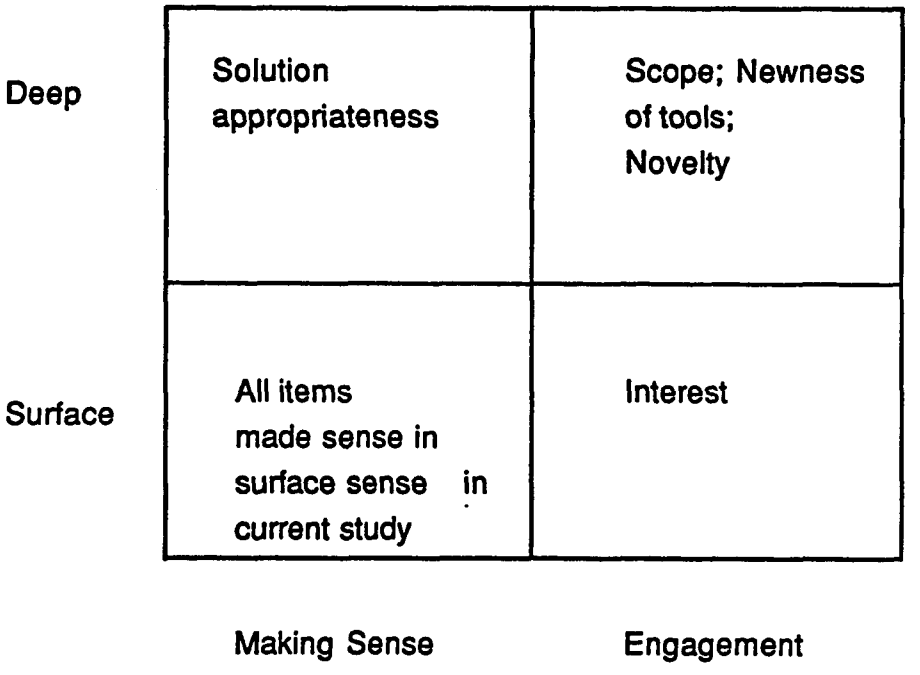


Figure 5.1

Preference Framework for Experts

science forms, each factor seems to have been represented. Each aspect apparently played a different role in and had a special impact on expert preference.

Summary

The preference ratings which were collected in this study can be thought of as indicators of perception. In this chapter, those ratings were grouped together statistically for the expert subject group. The resulting four categories may be indicative of the groupings which these subjects utilize in perception.

The four categories which emerged from this analysis were distinguishable in at least two major ways. On the one hand, they differed in the scope and novelty of the problems and tools which were presented in the items which were members of the categories. Secondly, the appropriateness of a solution for a presented problem seemed to distinguish the categories.

These issues of scope and novelty, and appropriateness bear some resemblence to salient issues in environmental preference. Because of this similarity in issues, it was possible to develop a model of expert computer science preference, based on the environmental preference framework. The exercise of developing the computer science framework suggested further ways in which expert problem representations were involved in the collected responses.

Notes to Chapter V

[1] Factor analysis refers to a family of statistical techniques which are widely used in behavioral research to extract underlying categories or dimensions out of a large data set. In general, factor analysis consists of three steps: generation of an interrelation matrix, extraction of initial factors, and rotation to a final factor structure. In the first step, the interrelationships among variables or individuals, expressed typically as correlations or covariances, are generated pairwise. From the resulting matrix of interrelationships, a set of initial factors is extracted. The initial factors may incorporate *a priori* assumptions about underlying regularities. Each initial factor describes a linear combination of the original variables or individuals. The initial factors are then rotated to a final solution. The final solution forms an n-dimensional structure. Each dimension typically contains one or more of the original variables or individuals. Category membership is determined by the "loadings" of individual items on the factors. The final rotation highlights high factor loadings. The original variables or individuals which are grouped in the same dimension have thus been shown to have some underlying common theme (Harman, 1976; Child, 1973).

The Guttman-Lingoes Smallest Space Analysis (SSA-III) factor analysis with a varimax rotation was performed using the data in this study. SSA-III is a non-metric factor analysis; the algorithm is non-metric in the sense that the correlation matrix is transformed into a rank-order matrix (Lingoes,1972).

In other studies of preference outside of computer science, this particular factor analysis technique has been used successfully to extract categories. Its proponents have contended that the resulting solutions tend to be more stable than those from other factor analysis techniques (cf. Kaplan,1975).

[2] Personal communication on August 1, 1986 with Mark Weaver, Rik Belew, John Vidolich, Kevin Ross, and Steve Kaplan, Ann Arbor, MI.

CHAPTER VI

PERCEPTUAL CATEGORIES: NOVICES

The previous chapter suggested that the experts in this study seemed to perceive items as similar (as members of categories), based on the scope and novelty of the problem, as well as the appropriateness of the solutions. In this chapter, the perceptions of novices, through their preferences, are examined. In particular, the patterns of these perceptions will be highlighted. Once again, the analysis is based on a SSA-III factor analysis, using the same criteria as in the previous chapter.

Recall that the results in Chapter IV indicated that different aspects of the test instrument items influenced the preference responses of experts and novices. In particular, experts tended to be more appreciative of problem characteristics and novices seemed to highlight the solution characteristics. (Variables of interest, solution classicalness, and discourse rule violation status influenced novice preference.) In this chapter, the factors which describe patterns of novice preference will be interpreted in large part in terms of solution characteristics.

Novice Factor Structure

Based on an SSA-III analysis of the novice preference ratings, five factors described the task items. Eighteen of the 20 items were included in the structure. The five categories were: the MODELS OF BUBBLE SORTS Category, the MODELS OF MAIN PROCEDURES Category, the GAMELIKE Category, the INTRODUCTORY PROGRAMMING Category, and the DIFFICULT Category. Table 6.1 provides the names of the items within each category, their loadings,

FACTORS	LOADINGS	SOLUTION CLASS.	PROB. CLASS.	DISCOURSE RULE STATUS	MEAN INTEREST
1. BUBBLESORT		100% not low	100%high	50% violations	3.01
Price List	-.6671	not low	high	1 or more	3.13
Contrib.	-.9590	not low	high	0	2.89
2. MAIN PROCS.		100% not low	0%high	50% violations	2.82
ThunderSt.	.7176	not low	not high	1 or more	3.04
Pet Poison	.7427	not low	not high	0	2.60
3. GAMELIKE		57% not low	42%high	71% violations	3.00
Grades	-.4637	not low	not high	0	3.35
Jungle Esc.	-.5445	low	not high	1 or more	2.89
Dog Weights	-.5426	not low	high	0	2.85
Rare Flora.	-.5146	not low	high	1 or more	2.55
Shoes	-.3959	not low	not high	1 or more	2.81
Toxic Sub.s	-.3884	low	high	1 or more	3.32
Prefab H.	-.5230	low	not high	1 or more	3.08
4. INTRO.PROG		67% not low	33% high	100% violations	2.94
JewelryStore	-.6175	low	high	1 or more	2.96
South P.	-.7741	not low	not high	1 or more	2.85
Secret Code	-.3795	not low	not high	1 or more	3.00
5. DIFFICULT		50% not low	50%high	100% violations	2.77
Legis. Vote	-.4842	low	not high	1 or more	2.58
Seawater	-.6090	not low	high	1 or more	2.54
Lottery	-.6362	low	not high	1 or more	3.09
Archeology	-.4628	not low	high	1 or more	2.87

Table 6.1

NOVICE FACTOR CHARACTERISTICS

and information on their levels of the variables of interest, classicalness, and discourse rule violation status. The next five sections describe each of the categories in detail. Appendix G contains the details of the factor analysis.

MODELS OF BUBBLE SORTS and MODELS OF MAIN PROCEDURES Categories

Two of the categories which emerged contained solutions which were similar to textbook models of algorithms. In particular, one category contained two bubble sorting algorithms, and the other category included modular main procedures. In each case, one item had discourse rule violations and one did not. In both groups, all of the items are of medium or high solution classicalness. The mean interest value for the items in the MODELS OF BUBBLE SORTS category is relatively high. The interest level for the items in the MODELS OF MAIN PROCEDURES category is moderate to low.

Several novices commented, after completing the experiment, that they noticed the solutions which were "just like they would do it." The bubble sorts in the MODELS OF BUBBLE SORTS category apparently seem to reflect the way that this group of subjects felt that they would implement a bubble sort. Not surprisingly, this would be the way that they have learned to implement this sort in their coursework. Similarly, the two main procedures which were included in the MODELS OF MAIN PROCEDURES categories would appear to reflect the way that the novices would expect to implement a main procedure.

GAMELIKE Category

This category includes seven items: two one-dimensional matrix rotates and one each of iterative procedure, insertion sort; binary table search with parallel arrays, detailed main procedure, exhaustive table search with parallel arrays. Slighty more than half of the items have high or medium solution classicalness levels. Five of the seven items in the group include one or more discourse rule violations. The mean interest score for the items in the group is relatively high.

The common feature among these solutions seems to be a gamelike quality. It is likely that the novices in this study would have had little previous exposure to these solutions. Yet the solutions should have been understandable to the subjects in the group. The items in this group presumably are challenging to the group because of their novelty, but still coherent because of the subjects' general familiarity with the constituent parts and their organization. The common feature of the components in this category, then, seems to be that they are relatively unfamiliar but easy to understand; that is, they are like games.

INTRODUCTORY PROGRAMMING Category

The three items in this group include a two-dimensional matrix rotation, an exhaustive table search with records, and an iterative procedure. The larger context for the solutions are Antarctic navigation, jewelry store inventory maintenance, and generation of a secret code, respectively. The interest level for items in this group are moderate. Two out of the three items have medium or high solution classicalness, but all of the items have at least one discourse rule violation.

Like the items in the expert's INTRODUCTORY PROGRAMMING category, these items are similar to typical introductory programming items. The solutions resemble pedagogic items which are developed for the sole purpose of illustrating programming concepts. The discourse rule violations are examples of such concepts. The uniftying theme would appear to be that this group contains items with contrived and pointless solutions

DIFFICULT Category

The last novice category contains four items, including both of the recursive solutions from the test instrument. The group also includes an insertion sort and a binary table search with records. Half of the items in this group contain solutions of medium or high classicalness, but each of the items contains at least one discourse rule violation. The interest level for the items in this group is low.

The novices in this study probably found the items in this group to be

difficult. It is well-known in the folklore of computer science education that students find recursion to be an extremely difficult concept to understand (cf. also Mynatt, 1984). Neither of the concepts presented in the other two items would have likely been a point of emphasis in the course work of the novice subjects, in spite of their relative commonality in computer science, in general. These items would seem to share primarily a high difficulty level for the novice subjects.

Relation of Categories to Preference

The means for preference are compared between categories in Table 6.2. As Table 6.2 illustrates, the categories MODELS OF BUBBLE SORTS and MODELS OF MAIN PROCEDURES have the highest mean preference scores, followed by the GAMELIKE category, the INTRODUCTORY PROGRAMMING category, and the DIFFICULT category. The differences among the category means is significant ($F(4,272) = 4.95$, $p < .001$).

Novice Categories and the Environmental Preference Framework

In the previous chapter, the environmental preference framework which was presented in Chapter II, provided a starting point for a model of expert preference in the computer science setting. The expert framework of the preceding chapter expressed the components of making sense and engagement in both an immediate and a future context. In this section, the results of this study for the novice subject group are compared to the environmental preference framework.

Making Sense and Engagement for Computer Science Novices

The categories of the novice dimensional structure seem to be conceptually distinguishable by at least one issue, namely familiarity. The items in the MODELS OF BUBBLE SORTS and MODELS OF MAIN PROCEDURES categories would seem to have high familiarity for the novice group of subjects since these items resemble common textbook and classroom examples. The items in the GAMELIKE and INTRODUCTORY PROGRAMMING categories are

CATEGORY	# OF ITEMS	PREFERENCE
BUBBLE SORTS	2	3.33
MAIN PROCS.	2	3.33
GAMELIKE	7	3.13
INTRO. PROG.	3	3.04
DIFFICULT	4	2.82

Table 6.2

Mean Preference Levels for
Novice Categories

likely to be of only moderate familiarity to the novice subjects. These items incorporate solutions which may have been seen before by this group of subjects, but not items which would have been figural. Low familiarity appears to be one of the issues which makes the elements in the DIFFICULT category more difficult to work with than the other items in the study.

The levels of novice preference judgements seem to parallel these issues of familiarity. For items which are perceived as being highly familiar (the elements in the MODELS OF BUBBLE SORTS and MODELS OF MAIN PROCEDURES categories), the preference level tend to be relatively high. Items which are not familiar (the items in the DIFFICULT category) tend to receive lower preference ratings. Items with intermediate levels of familiarity for the subjects (members of the GAMELIKE and INTRODUCTORY PROGRAMMING categories) tend to have moderate preference levels. One can guess that the novices in this study preferred items which were more familiar. Additionally, items which resembled each other in terms of familiarity level were perceived as being similar in some respects.

This issue of familiarity seemed to be involved in two of the independent variables which effectively predicted novice preference. The items which appear to be highly familiar (elements in the MODELS OF BUBBLE SORTS categories and MODELS OF MAIN PROCEDURE categories) also uniformly have high or medium solution classicalness levels. Items which seem to be less familiar (elements of the GAMELIKE, INTRODUCTORY PROGRAMMING, and DIFFICULT categories) tend also to have low solution classicalness levels. The highly familiar items also have fewer discourse rule violations than the items that are only moderately or slightly familiar.

The making sense component of the environmental preference framework seems to be closely related to the issue of familiarity in the context of novice computer science preference. In the environmental preference framework, settings which make sense are those in which the pieces fit together in meaningful ways. Within the computer science arena, for the novices sampled in this study, it would appear that items which are more familiar are more likely to

have pieces which fit together in meaningful ways.

A second issue, namely challenge, seems to also distinguish the categories in which the familiarity level of the items is moderate. For both the GAMELIKE and the INTRODUCTORY PROGRAMMING categories, the items are simple enough to be solvable by the novices. The difference between the groups appears to be in terms of challenge. The items in the GAMELIKE category seem to be more challenging, as reflected by their gamelike quality. By contrast, the items in the INTRODUCTORY PROGRAMMING category would seem to offer little challenge, since they bear such close resemblences to classroom materials.

The independent variable of interest seems to have been an expression of challenge for the novice subjects. The items in the GAMELIKE category generally received high interest ratings, while the interest ratings for the elements of the INTRODUCTORY PROGRAMMING category tend to be lower (although not significantly).

It has been suggested that the emergent novice categories and their characteristic preference levels parallel the reaction of a novice to items on a test (see Note 1). Students certainly appreciate test items which are directly from the textbook and class lectures. The items in the MODELS OF BUBBLE SORTS and MODELS OF MAIN PROCEDURES categories fit this description and have high preference ratings. Students also seem to enjoy test items which are challenging but solvable. Again, the items in the GAMELIKE category seem to match this description.

Students do not usually like problems of only moderate familiarity and no redeeming challenge. This description is appropriate for the items in the INTRODUCTORY PROGRAMMING category. Not surprisingly, the preference ratings for items in this group are only moderately high. Of course, students do not appreciate problems which are beyond their abilities. This is an apt description of the problems in the DIFFICULT category; the preference ratings for items in this group are low.

In summary, then, preferences of the novices in this study seemed to be affected by both familiarity and challenge. Items which were familiar or

challenging tended to receive higher preference ratings than items with less of these qualities. This pattern is similar to preference in the environmental context where higher levels of making sense and engagement lead to higher preference levels.

Novice Preference in an Immediate and a Future Sense

In the environmental preference framework, the components of making sense and engagement have meaningful descriptions in both an immediate and future context. The indicators for the present reflect surface level features of scenes, and the indicators for the future reflect deep level features. Expert preferences in this study followed a similar pattern. The components of both making sense and engagement for that group were related to both surface and deep structure attributes of the items. Is that pattern, found in both environmental preference and expert computer science preference, repeated for the novices in this study?

To answer a similar question for the experts in this study, characteristics of the subjects' problem representations were considered. In the same way, the problem representations of the novices will help to answer the current question.

Recall that novices do not invest a large amount of effort in the formation of an adequate problem representation. Instead they seem to briefly consider the problem and then move quickly to the generation of a solution. In the current study, where both the problem and solution were presented, this process was likely modified to be a brief consideration of the problem followed by both a brief and possibly a detailed consideration of the presented solution. Of course these processes are highly dependent on perception (which in this study was captured in part by preference ratings). During the consideration of the problem and the initial consideration of the solution, perception of surface level features occurs. If the subjects further consider the solution, they most likely extract more deeply embedded information from the items.

The independent variables which affected novice preference differed in their embeddedness into the items (this is shown in Figure 4.1). Interest,

according to the analysis in Chapter IV, is a measured variable which is related to the surface level features of the problem. One can speculate that the measured interest levels reflect something of the novices' quick consideration of the problem. That is, it is a surface level component. Interest also appears to be tied to the environmental preference component of engagement since it seems to reveal something about how challenging the problem is perceived to be. One can infer that interest for the novices, as well as for the experts, represents a surface level manifestation of engagement.

Discourse rule violation status is related to the use of a programming language. As such, it is probably an immediate feature of solutions. A novice who is considering a solution in a preliminary way would be likely to notice violations of the discourse rules. Because discourse rule violation status also appears to be a component of making sense, it seems reasonable to suppose that the variable is a surface level manifestation of that component.

The variable of solution classicalness was initially presented as being an embedded feature of the items. Since solution classicalness is a more subtle feature of the solutions than the more overt discourse rules, it seems unlikely that the variable would affect initial perceptions. It seems more likely that if and when novices considered the presented solutions in any detail, their perceptions might be affected by the level of solution classicalness. When the level of solution classicalness was low, this detailed consideration would be made more difficult than the situation of medium or high solution classicalness. Since solution classicalness was previously associated with making sense, it now seems appropriate to suggest that the variable is linked to making sense in a deep sense.

No variable measured in this study seemed to be related to engagement in a deep sense for this group of subjects.

For the novice subject group, then, three of the four facets of the environmental preference framework seem to have emerged. The matrix in Figure 6.1 shows a proposed model of making sense and engagement for the novices. Making sense is described in both a present and future context, while

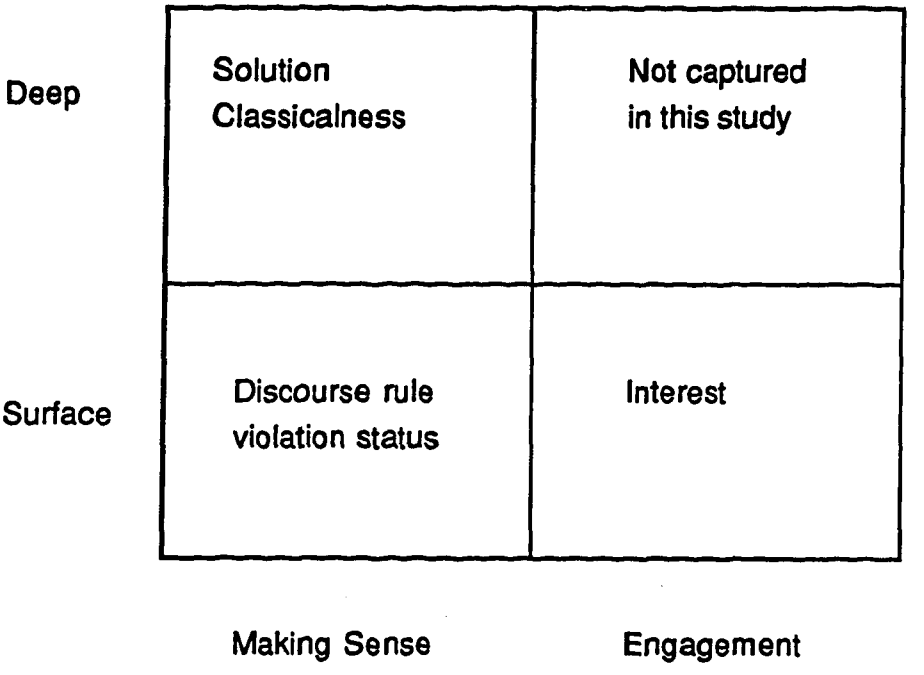


Figure 6.1

Preference Framework for Novices

engagement is expressed only in an immediate sense in this study.

Summary

The pattern of preference ratings of the novices can be thought of as a reflection of perception. In this chapter, the ratings were grouped together statistically. The five categories which emerged may reflect to some degree the perceptual categories which are used by these novices.

The five categories were distinguishable in two ways. On the one hand, items which resembled each other in terms of familiarity were grouped together. Also items which were similarly challenging tended to be aggregated.

The issues of familiarity and challenge are similar in some ways to the issues of making sense and engagement from the environmental preference framework. These similarities supported the development of a framework of novice preference. As the novice framework was developed, issues of problem representation were explored.

Notes to Chapter VI

[1] Private conversation on August 1, 1986 with Mark Weaver, Rik Belew, John Vidolich, Kevin Ross, and Steve Kaplan, Ann Arbor, MI.

[2] The structure of the dimensions for both experts and novices seems to have been influenced by both the generic item type and the particular content of the items in the category. For example, the experts' NOVEL USE OF A COMMON CONSTRUCT category included three matrix rotation problems. A fourth matrix rotation problem was included in the test instrument but was not included in this group or in any other dimension. The problem involved the rotation of prefabricated houses in a new housing development. This problem seems difficult to understand and somewhat contrived; the rationale for the rotation in the problem may have been problematic for the subjects.

The OFF OF THE SHELF category for the experts included two sorting problems and one searching problem. But another sorting problem appeared with the items in the INTRODUCTORY PROGRAMMING category. One an infer that the third sorting problem, which involved daily dog weights, seemed so contrived that it could not be considered as a real-world problem at all. The content excluded it from the OFF OF THE SHELF category.

The interplay between content and generic item type affected the membership in the novice categories as well. The two categories of MODELS OF BUBBLE SORTS and MODELS OF MAIN PROCEDURES clearly reflected the generic solution types of the items in the categories. However an iterative procedure appears in both the GAMELIKE and INTRODUCTORY PROGRAMMING categories. The item in the GAMELIKE category involved the summation of student grades; the item in the INTRODUCTORY PROGRAMMING category required the generation of a secret code. While the specific computational task involved is slightly different, the items both require a small numerical calculation. From the description, the secret code task would seem to be more gamelike, and the calculation of grades would seem more mundane. Yet to the novice students, secret code generation is contrived and grade calculation is gamelike.

CHAPTER VII

REVIEW, CAUTIOUS APPLICATIONS, AND CONCLUSIONS

In this section, the results of this study are reviewed. Following this review, the themes which emerged from these results are discussed together. Some applications for the results are cautiously offered.

Review of Results

In previous chapters, the preference ratings were analyzed in two ways. Each type of analysis had a different goal. The results of these analyses provided two different ways to understand the data.

The goal of the first type of analysis was to gain insights into the relationships between the dependent variable of preference and the independent variables of interest, problem classicalness, solution classicalness, and discourse rule violation status. In order to better understand these relationships, the independent variables were organized into a 2x2 matrix. In that matrix, the variables were described as either problem or solution variables (Interest and problem classicalness were problem variables, and discourse rule violation status and solution classicalness were solution variables). Also, within the matrix, the variables were described as either deep level, abstract or surface level variables (Problem and solution classicalness were deep level variables, and interest and discourse rule violation status were surface level variables).

The analysis technique which was used was analysis of covariance. This analysis was performed separately for the two groups of subjects.

For the expert subjects, the variables of problem classicalness, solution classicalness, and interest emerged as important. For the novice subjects, the

variables of solution classicalness, interest, and discourse rule violation status were significantly related to preference. In terms of the matrix of independent variables, these findings suggested that with increasing expertise, computer scientists tend to shift their focus. For the subjects with lower experience levels, the surface level variables of interest and discourse rule violation status were important. With increasingly greater expertise levels, the discourse rule violation status loses significance, but the abstract variable of problem classicalness gains importance. Similarly for the novice subjects, the variables which relate to the solution (solution classicalness and discourse rule violation status) were important. However with increasing expertise, the trend seemed to be to emphasize problem features. Again the solution variable of discourse rule violation status became less important, while the problem variable of problem classicalness became more figural.

The goal of the second type of analysis was to learn about the perceptual categories which were employed by the subjects. This was possible in the current study because preference is, in some sense, an indication of perception. In order to extract categories from the data, an SSA-III non-metric factor analysis was performed on the preference ratings. Because the results of the analyses of covariance indicated that the variables which are emphasized seemed to shift with increasing expertise, two SSA-III analyses were performed. One was conducted with the experts' responses, and the second was performed with the novices' responses.

For the expert subjects, four categories of responses emerged, including the UNIQUE category, the NOVEL USE OF A COMMON CONSTRUCT category, the OFF OF THE SHELF category, and the INTRODUCTORY PROGRAMMING category. These categories were distinguishable by the scope, real-worldness, and novelty of the problems and the tools to solve the problems. They were also differentiated by the appropriateness of the solutions for the problems. In terms of preference, items which had more extensive slope or novelty and appropriate solutions tended to receive higher preference scores than items which were less strong in these areas.

The independent variables seemed to reflect these issues of scope, novelty, and appropriateness of solutions. In particular, items which had high levels of problem classicalness appeared to generally have more limited scope than items with lower problem classicalness levels. Items with high interest levels seemed to have broader scope than items with lower interest levels. Finally, items with unexpected solutions which were paired with relatively conventional problems resulted in a perception of inappropriateness in some cases.

For the novices, five categories emerged, including the MODELS OF BUBBLE SORTS category, MODELS OF MAIN PROCEDURES category, GAMELIKE category, INTRODUCTORY PROGRAMMING category, and the DIFFICULT category. These categories seem to be distinguishable by both the familiarity and challenge level of their items. In terms of preference, items which were familiar and challenging tended to be more highly preferred than items which were less strongly representative of these attributes.

The independent variables which were significant to the novices (solution classicalness, discourse rule violation status, and interest) also seemed to be related to familiarity and challenge. In particular, items which included solutions of medium or high classicalness or contained no discourse rule violations tended to enhance familiarity. Items with high interest levels appeared to have been perceived as being more challenging than items with low interest levels.

Review of Themes

Two themes, involving the roles of problem representation and preference, seem to have surfaced in this study. In this section, the two issues are first discussed separately and then together.

Problem Representation

The analyses of covariance in particular seem to indicate that growing expertise is accompanied by shifts in importance of some attributes of the items. While problem and solution features were important to both of the subject groups,

the results of the analyses suggested that with increasing expertise the problem components seem to become generally more important than the solution features. Similarly deep and surface level features are important to both subject groups. However, with increasing expertise there appears to be a shift in emphasis away from the surface level features toward emphasis on deep level features.

One theme that emerges from the body of expertise literature, both within and outside of computer science, is that much of the power of the expert lies in her ability to form a superior problem representation. The expert, as it turns out, focuses much of her problem solving activity on the formation of such a representation and little energy on the actual generation of a solution. Novices, by contrast tend to concentrate on the solution. The formation of an adequate problem representation involves both the formation of an initial representation and refinement of the representation. Since both processes involve considerable interaction with the environment, the expert relies on perceptual skills throughout.

The results from the analysis of covariance suggest that differences in problem representations were involved in the differences between the experts and novices in this study. The experts tended to emphasize problem characteristics more than solution features and the novices highlighted solution characteristics. One can infer that the experts focused on problem issues as they developed their problem representations. In spite of the presented solution, the experts seemed to solve the problem (that is, formed a problem representation) as if the solution was not even present. The novices in this study, as they have done in previous studies, move quickly to the solution.

Preference

In the domain of environmental preference a framework, based on information, has emerged. In this framework, preference is accounted for by two components: engagement and making sense. Each of these components is meaningful in both an immediate and a future context. Some aspects of the

preference categories for both the expert and novice subject groups appear to be related to aspects of the informationally-oriented environmental preference framework.

Expert Preference - For the expert subjects, some patterns of preference seemed to be related to each of the facets of the environmental preference framework. The making sense component of the environmental preference framework was related to expert preference in both an immediate and future sense. For this group, it seemed to be directly related to the goodness of fit of the pieces of the items. Since the items were all relatively simple for this group, the pieces of all of the items seemed to fit together, suggesting that they made sense in a surface level way. For this group, making sense in a deep context was affected by the appropriateness of the solutions for the problems.

The engagement component of the environmental preference framework was also reflected in the experts' preference patterns in both a present and future sense. This component appeared to have been expressed in terms of issues of novelty and scope. Novelty and scope were expressed to some degee in interest levels in an immediate sense, and by problem classicalness in a deep sense.

Novice Preference - The patterns of novice preference in this study seemed to be related to three of the four facets of the environmental preference framework. The making sense component of the environmental preference framework was related to novice preference in both an immediate and future sense. For this group, degree of familiarity appeared to be an indicator of making sense. Lack of discourse rule violations seemed to enhance making sense in an immediate sense, while high or medium solution classicalness boosted making sense at a deep level.

The engagement component of the environmental preference framework was also reflected in the novices' preference patterns, but only in an immediate sense. Challenge, as reflected by interest, was interpreted to be an indication of

engagement in an immediate sense.

Problem Representation and Preference

Both the experts and novices in this study appeared to have characteristic patterns of problem representations and of perceptions, as reflected by their preferences. By considering what happens during the formation of problem representations for these subjects, one can begin to see how their perceptions are related.

For experts, the pattern of problem representation formation appears to be to form an initial representation of the problem, followed by a refinement of that representation. The formation of the initial representation involves a quick glimpse of the problem. This quick glimpse is enhanced if all of the pieces of the stimuli fit together readily. For this group of subjects, since all of the items in the study seemed to have been perceived to make sense initially, it seems reasonable to infer that initial representations of all of the items were formed.

If the stimuli initially seem to promise that future discoveries are possible with further investigation, then the process of refinement has the potential for payoff. For these experts, items which were "eye-catching," as reflected by high interest levels, seemed to have been perceived as holding this promise.

For the experts in this study, the refinement process appears to have been enhanced by the items which were perceived to continue to fit together, through repeated review. In particular, items in which the specified solution seemed appropriate for the problem appeared to fit this description. The refinement also appears to have been enhanced by items which were perceived to be incomplete and to offer opportunities for further learning. In this study, for the experts, the items whose situations were of broad scope or whose problems included high degrees of novelty seemed to enhance the refinement process in this way.

For novices, the process of forming a representation seems to involve a different sequence of events. In particular, their pattern of formation would seem to be to form an initial representation of the problem and a limited number of solutions (which in this study is the one presented solution), followed by some

additional consideration of the solution.

In this study, for the novice subjects, formation of an initial representation of the problem and one solution seemed to have been enhanced if the subjects perceived the problem as challenging, and the solution as familiar, in a surface level sense. If any further refinement of their representations occurred, it was facilitated by solutions which continued to be perceived as familiar, after repeated scrutiny.

Some Cautious Applications

The results of this study suggest that, within the limited cross section of computer science sampled, perceptions, as reflected by preferences do in fact change with increasing expertise. Moreover problem representations and their formation, which depend on perception, take on different forms with increasing expertise. Is it possible to apply these results to any real domains in computer science?

At least one area of computer science would seem to be able to benefit from the results of this study. Computer science education, by its nature, involves novices and experts. For example, for an expert who is teaching a class of novices, similar to the novices sampled in this study, the behavior of the novices may seem to be bizarre. Consider the following scenario:

> An experienced computer science educator has just completed a lecture about an important algorithm in computer science (such as quicksort) to a group of students, similar to the novices in this study. At least one novice at this level will invariably ask about the semi-colon in line 2 or line 3, etc.

This author, having experienced this scenario many times, has often wondered if the student who asked the question was asleep during the lecture and simply noticed the semi-colon as he or she was waking up. Unfortunately, this explanation is rarely adequate, since this type of apparently inappropriate question often comes from conscientious and alert students.

The results of this study suggest that given a problem (sorting), an

algorithm (quicksort), and some program code (including the questionable semi-colon), novices will tend to focus on the solution, particularly the surface level issue of the semi-colon. The expert (in this case, the author), by contrast, is concentrating on the problem and the abstract solution features (ie the algorithm) and virtually ignoring the surface level solution features, such as the semi-colon. In other words, the expert and novices in this scenario seem to perceive the presented material differently.

This author, like other computer science educators, has noticed many other instances of apparently bizarre student behavior (A few colorful examples include interest in packed arrays of records, interest in base minus two arithmetic, and a program named MongolianYakHerder with the programming variables named after yaks.). One possible application of this study might be to enhance the awareness of computer science educators that novice perceptions are different. These perceptions are not really bizarre, but are consistent with and provide important clues into the ways that the novices structure their computer science world.

This study also revealed something about the characteristics of computer science entities which lead to higher preference levels. For example, the results suggested that for novice programmers, settings which are both familiar and challenging, reflected by interestingness, tend to be preferred. For computer science educators, this suggests that the programming examples and assignments, which form the basis for programming education, should include some degree of each of these components. Given a choice between two otherwise identical assignments or examples, the one which is more highly endowed with these attributes is likely to be preferred. So, for example, given the choice of an assignment to implement the game of Battleship (a familiar game but a challenging program) or Gorilla Rescue (a game invented by this author which is the game of Battleship in the disguise of a game about rescuing mountain gorillas from poachers), the students may be more likely to prefer Battleship. (This is supported by this author's experience.)

A second application of this study and its results arises from the

methodology. In the current study, the methodology involved a series of twenty "software development snapshots." A software development snapshot consisted of a brief description of a software engineering situation, statement of specific software development problem, and a partial program solution to the problem, each presented on one page. Unlike other types of stimuli which are often used in behavioral experiments in computer science, the snapshots were both short enough to be managable in an experimental setting and rich enough to preserve some aspects of real software engineering situations. That this methodology produced meaningful results suggests that it likely may have further application in other studies of human-computer interactions.

Conclusion

A key motivation in undertaking a study of this sort was to develop more than a folklorish approach to aesthetics and its interaction with expertise within computer science. The study was conceived under two theoretical premises: 1. Characteristics of the problem representation are closely tied to expertise. This is a general phenomenon which is associated with many facets of problem solving, including the formation of preference judgements. 2. The environmental preference model, due to the Kaplans (1982) may have applicability to domains aside from the environmentally-oriented ones, since it is based on the structuring of information rather than specific content.

Both of these theoretical notions were expressed in the results of this study. Increasingly sophisticated problem representations seem to be characteristic of experts in this study, as indicated by increasing focus on problems and solutions, even in the presence of presented solutions. As expertise level changes, the perceptions which underlie problem representations change as well. Moreover, the environmental preference framework provided a meaningful context for the patterns of preferences of both the experts and novices in this study.

APPENDICES

APPENDIX A

UNSUCCESSFUL ANALYSES

The items in the test instrument were systematically varied across variables of problem classicalness, solution classicalness, and discourse rule violations. Additionally, demographic variables of expertise level and gender were collected. Expertise level, problem classicalness, and solution classicalness were meaningfully included in the multiple regression models. Discourse rule violations were also included in that model; however the model incorporated only the presence or absence of one or more violations. This section reports on analyses based on gender, and on specific variations of the discourse rule violations.

Role of Gender

The mean-per-item preference scores were calculated for male and female subject groups. An analysis of variance was performed, with the 40 mean-per-item preference ratings as the dependent variable and gender as the independent variable. No significant preference differences, due to gender were found.

Specific Discourse Rule Violations

For both the double duty rule and the construct affordances rules, mean category scores for both the violate and no violate condition were calculated for each subject. For each discourse rule, a two-way analysis of variance was performed. The independent variables were category membership and expertise level; the dependent variable was the set of category scores. Significant category effects were found only for the double duty rule ($F(90,1)=22.29$, $p<.0001$); no significant effects were found for category membership for the construct affordances rule, expertise level for either rule, or interactions for either rule.

The effect of the discourse rules was also tested between pairs of functionally equivalent solutions of the same type, in which one member of the pair had a violation of the rule. Two-way analyses of variance were conducted on the pairs. The independent variables were discourse rule violation status and expertise level; the dependent variables were the preference ratings for the items in the pair. Four comparisons were possible for the double duty rule; ten comparisons were possible for the construct affordances rule.

For the double duty rule, one significant effect occurred, due to the violation status of the double duty rule. The difference was between two main procedures with high levels of detail ($F(90,1)=6.12$, $p < .02$). Higher preference ratings were given in the non-violation case.

In four of the comparisons of the construct affordances rule, a significant effect occurred. In all cases, the significant effect was due to the violation level of the construct affordances rule. No significant effects due to expertise level or interaction between expertise and the violate condition occurred. The results of the analysis of the discourse rules is shown in Table A.1.

In both the pair of exhaustive searches and the pair of detailed main procedures, the WHILE construct is preferred over the REPEAT-UNTIL construct; the WHILE is appropriate in the search items and inappropriate in the main procedures.

The results of these two pairwise comparisons involving the WHILE construct, which showed significant differences, seem to suggest that the WHILE may be preferred instead of the REPEAT, even when it is not appropriate in the strictest sense. However, this iterpretation must be made cautiously; two other pairs of items which involved REPEAT and WHILE showed no significant effects.

The pair of detailed main procedures was also included as a pair in which one item contained a double duty violation and one did not. In fact, the same item contains both types of violations. Given this confounding, it is impossible to know if the pairwise difference is the result of one, both or none of the discourse rules.

The other two pairs which showed significant differences involved

ANALYSIS	SIGNIFICANT	COMMENTS
Mean Category Scores		
1. Double Duty Rule		
Expertise	ns	
Violation Status	F(90,1) =22.29, p<.0001	
Interaction	ns	
2. Construct Affordance Rule		
Expertise	ns	
Violation Status	ns	
Interaction	ns	
Pairwise Comparisons - Double Duty Rule		
1. Jungle Escape-Arrange H.		
2. South Pole - Quilting	ns	
3. Weights - Seawater	ns	
Expertise	ns	
Violation Status		
Interaction		
4. Library System - Shoe S.		
Expertise	ns	
Violation Status	F(90,1) = 6.12, p<.02	no violation preferred
Interaction	ns	
Pairwise Comparisons - Construct Afford		
1. Jungle Escape - Arrange H.		
2. South Pole - Quilting		
3. Contributions - Price List		
4. Weights - Seawater		
5. Pet Poison Hotline - Thunder		
6. Archeology - Rare Wild Flora		
Expertise	ns	
Violation Status	ns	
Interaction	ns	
7. Library System - Shoe Store	see above	violation preferred
8. Jewelry Store - Toxic Subs	F(90,1) = 16.48, p<.0001	no violation preferred
9. Secret Code-Grades	F(90,1) = 7.52, p<.01	violation preferred
10. Lottery Number-Legis. Vote	F(89,1)=17.61, p<.0001	no violation referred
expertise, interaction	ns	

Table A.1- Details of Discourse Rule Violations

numerical calculation problems. In one case, the violation condition (procedure used instead of a function) was preferred. In the other case, the non-violation condition (function used instead of procedure) was preferred. This apparent paradox makes these results difficult to interpret.

Comments

Soloway and Ehrlich's (1984) original study of the discourse rules showed significant performance differences between experts and novices, violation and no violation condition, and the interaction of expertise level and violation status. In particular, they found decreased performance when the double duty rule was violated or when the WHILE construct is substituted for the IF construct. In the pairs of items in the current study in which a WHILE construct was substituted for an IF construct, no significant differences occurred.

This noticable difference in results can be explained, at least in part, by difference in the stimuli in the two studies. Soloway and Ehrlich used programming solutions as their only stimuli; the task in the current study involved both problems and solutions. As the results of the previously-described regression analysis indicates, experts focus on the problem component of the task and novices focus on the solution. It is not surprising that the discourse rules did not significantly contribute to differences between novices and experts in the current study. Experts apparently did not consider the surface level characteristics of the solutions when they made preference judgements. Figure A.1 summarizes the differences between Soloway and Ehrlich (1984) and the current study.

Soloway and Ehrlich (1984) assumed that their performance measurements were valid indicators of what programmers liked; that is, their preferences. However, because this assumption has not been tested empirically, either in the current study or elsewhere, its validity remains unknown.

Similarities

Current Study	Soloway and Ehrlich (1984)
1. Presents incomplete item; Subject relies on contextual information	1. Presents incomplete item; Subject relies on contextual information

Differences

1. Collection of preference ratings	1. Collection of performance measurements
2. Presentation of problem and partial solution	2. Presentation of solution

Figure A.1 - Similarities and Differences between current study and Soloway and Ehrlich (1984)

APPENDIX B

SAMPLE TASK ITEM

The following pages (35) contain a sample of the description of the project and the task which were completed by the subjects in this experiment.

Perception of Software Quality

Project Description:

Within computer science, factors which describe "what programmers like in programs" are often considered to be idiosyncratic. Such factors are typically attributed to individual style.

The purpose of this project is to explore an alternate hypothesis. That alternative is that there is widespread similarity among the factors that describe "what programmers like in programs." A related goal is to identify some of the factors that predict what programmers like in programs.

Please do not discuss this experiment with anyone else.

If you have any questions about this project after we are through, please feel free to contact:

Laura Leventhal
Computer Science Department
Bowling Green State University
Bowling Green, Ohio 43403

419 - 372 -2765

DIRECTIONS

Task Description:

1. The following pages contain a set of problems for you to work with. Each problem is presented with solution.

2. The solution that is presented is a program segment. This means that the solution only shows those operations that are required to solve the problem. All other declarations or code that are indirectly related to the problem or the solution are not specified.

3. Each of the segments is syntactically and logically correct.

4. You will be asked for your opinion of each of the presented program solutions.

5. The problems and solutions should be self-explanatory; however, if you have any questions, at any time, feel free to ask the monitor.

6. THIS IS NOT A TEST OF YOUR PROGRAMMING ABILITY.

Directions:

-- Each of the problems is independent of any of the other problems. This means that you should not compare any of the problems and answers with any of the other problems and answers.

DO NOT GO BACK TO A PROBLEM THAT YOU HAVE ALREADY FINISHED WITH!! Once you answer a problem, GO ON TO THE NEXT ONE.

-- You may write on the test booklet.

-- If you have any questions at this point, please ask the monitor.

-- You may start when the monitor gives you the signal.

Thank you for you participation.

1. PROBLEM: A friend of yours has a dog who is overweight. The dog has been on a diet for several months. Your friend has been weighing his dog everyday.

Now your friend is curious about the range of weights that the dog has had. You agree to write a program that will sort the weights in ascending order and print a report.

The following code segment compiles the list and generates the report.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
.
.
.
for i := 1 to number_of_weights do
   begin
      wgt_index := 1;
      found := false;
      while (not found) and (wgt_index <= i) do
            if (weights [i] < update_weights [wgt_index])
                  then found := true
            else wgt_index := wgt_index + 1;
      for j := i downto wgt_index do
            update_weights [j + 1] := update_weights [j];
      update_weights [wgt_index] := weights [i]
    end;
print_weights;
.
.
.
```

Rate this program segment:

A = unsatisfactory
B = marginally acceptable
C = adequate
D = highly acceptable
E = elegant

2. PROBLEM: You are asked by the government of a small country to develop an encoding algorithm for their "number of the day". Their number of the day is used as a password into their computer system in which all of the data about their extensive national park system is stored.

The following procedure produces a secret code.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
.
.
.
procedure secret_code (seed, key: integer; seed_table: seed_type;
                       var day_code: integer);
  var i: integer;
    begin
        day_code := truly_random(seed);
        for i := 1 to key do
            day_code := day_code *
                        seed_table(truly_random(i))
    end;

.
.
.
```

Rate this program segment:

A = unsatisfactory
B = marginally acceptable
C = adequate
D = highly acceptable
E = elegant

3. PROBLEM: You are writing a program for the game, Jungle Escape. Jungle Escape is a board game for two players. In the game, the physical position of the player, in relation to the board, is a relevant parameter. In the computer version of the game, you must be able to rotate the board 90 degrees.

The following code segment accomplishes the required rotation.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
                .
                .
                .
j := 1;
while (i <= boardsize) do
      begin
            j := 1;
            while (j <= boardsize) do
               begin
                 offset := (boardsize * (j - 1)) +
                                 ((boardsize - i) + 1);
                 new_offset := ((i - 1) * boardsize) + j;
                 new_board [new_offset] :=
                           board [offset];
                      j := j + 1
               end;
            i := i + 1
      end;
copyboard (board, new_board);

                .
                .
                .
```

Rate this program segment:

A = unsatisfactory
B = marginally acceptable
C = adequate
D = highly acceptable
E = elegant

4. PROBLEM: You are asked by your state government to develop an algorithm to generate a lottery number. This lottery number will be used as the winning number for the weekly Jackpot Bonanza.

The following procedure produces a lottery number.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
.
.
.
procedure number_generator (seed, key: integer;
                            var jackpot_number: integer);
    begin
        if (key = 1) then
            jackpot_number := random_time(seed)
        else
            jackpot_number := seed *
                number_generator
                 (random_time(seed), (key -1),
                     jackpot_number)
    end;

.
.
.
```

Rate this program segment:

A = unsatisfactory
B = marginally acceptable
C = adequate
D = highly acceptable
E = elegant

5. PROBLEM: You have just been elected as the treasurer of your favorite charitable organization. Your first job is to sort the contributions from a recent telephone campaign and print a report. The sorted contributions should be in order from generous to most generous.

The following code segment compiles the list and generates a report.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
.
.
.
for i := 1 to number_of_contributions do
      for j := i to number_of_contributions do
            if (contribution [j] < contribution [i]) then
              begin
                  save := contribution [j];
                  contribution [j] := contribution [i];
                  contribution [i] := save
               end;
print_report;
      .
      .
      .
```

Rate this program segment:

A - unsatisfactory
B - marginally acceptable
C - adequate
D - highly acceptable
E - elegant

6. PROBLEM: A small jewelry store specializes in watch repairs. Not surprisingly, the store has a huge inventory of watch parts. Every day new parts are delivered. The owner and chief repair technician asks you to write an inventory program.

The following code segment updates the part catelogue.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
        .
        .
        .
read (delivery_item);
if (catelogue [1].part_number > 0) then
      begin
            updated := false;
            count := 2
      end
else updated := false;
while ((not updated) and (count <= catelogue [1].part_number))
    do
      if (item = catelogue [count].part_number) then
        begin
            catelogue [count].quantity :=
                  catelogue [count].quantity + 1;
                  updated := true
        end
      else count := count + 1

        .
        .
        .
```

Rate the program segment:

A = unsatisfactory
B = marginally acceptable
C = adequate
D = highly acceptable
E = elegant

7. PROBLEM: You have just been appointed as the Official Magistrate of Development by the City Council. Your primary duty as Magistrate is to construct neighborhoods out of pre-fabricated houses. You must place houses on the street so that the front door faces the street.

This is a tedious job so you appoint a computer salesman as your assistant and receive a free computer as compensation.

You and your assistant write a program that will take a grid, which represents a pre-fabricated house, and reorient the grid on any street appropriately.

The following code segment accomplishes the required reorientation.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
.
.
.
for i := 1 to house_size do
   for j := 1 to house_size do
      new_house [(house_size * (j - 1)) +
            ((house_size - i) + 1)] :=
                house [((i - 1) * house_size) + j];
display_house (house, new_house);
.
.
.
```

Rate this program segment:

A = unsatisfactory
B = marginally acceptable
C = adequate
D = highly acceptable
E = elegant

8. PROBLEM: Your next exam in your computer science class consists of true and false problems. Your instructor is exhausted and offers to assign you an "A" in the class if you will write a test grading program.

The program will input the test answers for each student. You are to compare the student's answers with a key. The student must have at least 70% of the answers right to pass the exam. For each exam which is input, you are to report "Pass" or "Fail."

The following function finds a raw score for a single exam.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
.
.
.
function grade (first_question, last_question: integer; exam, key:
                exam_type) : integer;
  var score, i : integer;
      begin
            score := 0;
            for i := first_question to last_question do
                  if exam [i] = key [i] then
                        score := score + 1;
            grade := score
      end;

.
.
.
```

Rate this program segment:

A = unsatisfactory
B = marginally acceptable
C = adequate
D = highly acceptable
E = elegant

9. PROBLEM: You are to write a main procedure for an on-line ordering program. The program will be used by the Technocrat On-Line Shoe Company. Since Technocrat wants to sell as many pairs of shoes as possible, the buyer is allowed to use the ordering system as many times as he or she would like.

The following code segment is the main procedure for the ordering program.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
.
.
.
begin
    writeln ('enter y to order again,
                anything else to quit');
    readln (answer);
    while (answer = 'y') do
      begin
        technocrat_profile;
        technocrat_match_by_fit;
        print_order_descript;
        writeln ('enter y to order again,
                    anything else to quit');
        readln (answer)
      end
end;
.
.
.
```

Rate this program segment:

A = unsatisfactory
B = marginally acceptable
C = adequate
D = highly acceptable
E = elegant

10. PROBLEM: Northwest Ohio contains a wide variety of historical artifacts. An archeologist has a grant from a local historical society to collect and catelogue some of these valuable artifacts. This archeologist has organized the artifacts by type.

You are asked by the archeologist to write a program that maintains an inventory of the dig.

The following code segment updates the artifact catelogue.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
found := true;
done := false;
read (artifact-type);
while ((catelogue[0].number > 0) and (not done))
      begin
            first := 1;
            last := catelogue [0].number;
            found := false;
            done := true
      end
while ((last >= first) and (not found)) do
      begin
            middle := (first + last) div 2;
            if (artifact_type < catelogue [middle].number)
                  then last := middle - 1
            else if (artifact_type > catelogue [middle].number)
                  then first := middle + 1
            else begin
                  catelogue [middle].quantity :=
                        catelogue [middle].quantity + 1;
                  found := true
                  end
      end
```

Rate this program segment:

A = unsatisfactory
B = marginally acceptable
C = adequate
D = highly acceptable
E = elegant

11. PROBLEM: You are to write a main procedure for a library reference system. This system helps overworked undergraduate students generate term papers on any subject. Since a student may have to write more than one paper, the user may invoke the system as many times as he or she would like.

The following code segment is the main procedure for the system.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
.
.
.
begin
    quit := false;
    repeat
        current_topic_situation;
        paper_generator;
        print_reference_results;
        writeln ('enter y to use again, n to quit');
        readln (answer);
        if (answer <> 'y') then
            quit := true
    until (quit)
end.
.
.
.
```

Rate this program segment:

A = unsatisfactory
B = marginally acceptable
C = adequate
D = highly acceptable
E = elegant

12. PROBLEM: The plants of the world are in trouble; many rare plants are facing extinction. A botonist has a grant from a local horicultural society to observe, count, and catelogue some of these rare plants. This botonist has organized the plants by type.

You are asked by the botonist to write a program that maintains an inventory of the plant findings.

The following code segment updates the plant catelogue.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
                .
                .

        read (plant_type);
        if (not empty) then
           begin
              first := 1;
              last := catelogue_size;
              found := false;
              while ((last >= first) and (not found)) do
                 begin
                    middle := (first + last) div 2;
                    if (plant_type < type_number [middle])
                             then last := middle - 1
                    else if (plant_type > type_number [middle])
                             then first := middle + 1
                    else begin
                             type_quantity [middle] :=
                                      type_quantity [middle] + 1;
                             found := true
                          end
           end
         end
        else;
```

Rate this program segment:

A - unsatisfactory
B - marginally acceptable
C - adequate
D - highly acceptable
E - elegant

13. PROBLEM: You have been working as the aquarium curator at the zoo. You installed a saltwater tank several months ago, and you have been importing seawater ever since. You would like to start making the seawater yourself, so you have been analyzing some of the imported samples.

You have noticed considerable variation in the magnesium concentration of the imported seawater. Now you would like to organize your concentration data. You write a program to sort the data and print a report.

The following code segment compiles the list and generates a report.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
                .
                .
                .
for i := 1 to number_of_concentrations do
  begin
     magnes_index := 0;
     repeat
          magnes_index := magnes_index + 1
     until ((orig_list [i] < update_list [magnes_index]) or
                    (magnes_index > i))
     for j := i downto magnes_index do
          update_list [j + 1] := update_list [j];
     update_list [magnes_index] := orig_list [i]
  end;
print_magnes_report;
                .
                .
                .
```

Rate this program segment:

A = unsatisfactory
B = marginally acceptable
C = adequate
D = highly acceptable
E = elegant

14. PROBLEM: It is 1911 and you are lucky enough to be the navigator on Roald Amundsen's expedition to the South Pole. You are ahead of your time; you have a small microcomputer.

The polar area is represented on a map as a square that is made up of small square segments.

The South Pole presents an interesting navigational challenge. No matter which way you move from the Pole, you are going north. Your navigational computer must be able to rotate the map to correspond to your position.

The following code segment accomplishes the required rotation.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
                    .
                    .
                    .
i := 1;
while (i <= side_size) do
      begin
            j := 1;
            while (j <= side_size) do
                  begin
                     offset := (side_size - i) + 1;
                     guide_map [i,j] :=
                         map [j, offset];
                     j := j + 1
                  end;
            i := i + 1
      end;
display_map (map, guide_map);
                    .
                    .
```

Rate this program segment:

A = unsatisfactory
B = marginally acceptable
C = adequate
D = highly acceptable
E = elegant

15. PROBLEM: You have recently developed an interest in making quilts. Because these quilts are labor-intensive and you do not want to make a mistake, you need a computer simulation of the quilt and the quilt-making process. One of the operations that your simulation must include is a rotation operation; this corresponds to rotating the quilt in its frame.

The following code segment accomplishes the required rotation.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
.
.
.
for i := 1 to quiltsize do
      for j := 1 to quiltsize do
            new_quilt [i,j] :=
                  quilt [j, ((quiltsize - i) + 1)];
copyquilt (quilt, new_quilt);
.
.
.
```

Rate this program segment:

A = unsatisfactory
B = marginally acceptable
C = adequate
D = highly acceptable
E = elegant

16. PROBLEM: Your company uses a wide variety of chemical substances; some of them are toxic.

Your company would like to keep a record of the number of employees who come into contact with a toxic substance in their normal work.

You are asked by the chairperson of the board to write a program that maintains an employee toxic substances inventory.

The following code segment updates the employee catelogue.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
.
.
.
read (substance_type);
if (number_substances > 0) then
  begin
    count := 1;
    repeat
      if (substance_type = type_number [count]) then
        substance_quantity [count] :=
            substance_quantity + 1
      else count := count + 1
    until (count > number_substances)
.
.
.
```

Rate the program segment:

A - unsatisfactory
B - marginally acceptable
C - adequate
D - highly acceptable
E - elegant

17. PROBLEM: Some friends of yours are having a four-family garage sale. These friends have cleaned out their basements and attics. Together, they have enough bargains to fill a two-car garage.

Now your friends want to place an ad for their sale in the newspaper. They want to print the prices in the ad, with the lowest prices first. You agree to write a program that will sort the prices in ascending order and print the list.

The following code segment compiles and generates the list.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
.
.
.
for i := 1 to number_of_bargains do
      for j := (i + 1) to number_of_bargains do
            while (bargains [j] < bargains [i]) do
               begin
                  temp_bargain := bargains [j];
                  bargains [j] := bargains [i];
                  bargains [i] := temp_bargain
               end;
print_bargains;

.
.
.
```

Rate this program segment:

A = unsatisfactory
B = marginally acceptable
C = adequate
D = highly acceptable
E = elegant

18. PROBLEM: Your state legislature is having a hard time evaluating the voting on bills. When the votes are tabulated by hand, a few votes usually seem to disappear.

The governor has allocated a special fund to pay for a computerized vote tabulation system, and has appointed you as the programmer.

Your program inputs a vote from each legislator on each bill. For each bill you are to tabulate the votes. Depending on the type of legislation, different numbers of "aye" votes are required to pass the bill.

The following function tabulates a raw score for a single bill.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
function tab_bill (first_voter, last_voter: integer;
                   bill_vote: tabulate_type): integer;
begin
    if (first_voter = last_voter) then
        if bill_vote [first_voter] = 'a' then
            tab_bill := 1
        else tab_bill := 0
    else
        tab_bill := tab_bill (first_voter,
                    (last_voter div 2), bill_vote) +
                tab_bill ((last_voter div 2 + 1),
                    last_voter, bill_vote)
end;
```

Rate this program segment:

A = unsatisfactory
B = marginally acceptable
C = adequate
D = highly acceptable
E = elegant

19. PROBLEM: You are to write a main procedure for the Pet Poison Hot Line. Since a pet may have swallowed many poisons, the pet owner is allowed to use the hot line as many times as he or she would like.

The following code segment is the main procedure for the hot line.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
        .
        .
        .
begin
    repeat
        current_pet_situation;
        diagnostic_symptoms;
        print_antidote
    until (finished)
end.

        .
        .
        .
```

Rate this program segment:

A = unsatisfactory
B = marginally acceptable
C = adequate
D = highly acceptable
E = elegant

20. PROBLEM: You are to write a main procedure for the On-Line Thunder Storm Locator for Pilots. Since a pilot may fly in different areas, the user of this system must be allowed to check as many areas as he or she would like.

The following code segment is the main procedure for the system.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
.
.
.
begin
      while (pilot_wants_to_know) do
        begin
            meteorological_scan;
            meteorological_predictor;
            display_storm_patterns
        end
end.
.
.
.
```

Rate this program segment:

A = unsatisfactory
B = marginally acceptable
C = adequate
D = highly acceptable
E = elegant

21. Your age is:
    A. Under 18
    B. 18 - 23
    C. 24 - 34
    D. 35 - 49
    E. Over 50

22. Your sex is:
    A. Female
    B. Male

23. Your major is:
    A. Pre - Computer Science; College of Arts and Sciences
    B. Computer Science
    C. College of Arts and Sciences; outside of Computer Science or Pre-Computer Science
    D. College of Business
    E. Other

24. Your grade point average overall is:
    A. 3.5 - 4.0
    B. 3.0 - 3.49
    C. 2.5 - 2.99
    D. 2.0 - 2.49
    E. Below 2.0

25. Your grade point average in your major: (if you have not undecided your major, skip this question)
    A. 3.5 - 4.0
    B. 3.0 - 3.49
    C. 2.5 - 2.99
    D. 2.0 - 2.49
    E. Below 2.0

26. Your class rank is:
    A. Freshman
    B. Sophomore
    C. Junior
    D. Senior
    E. Graduate student or post - baccalaurate

27. The number of college-level computer science courses that you have completed:
    A. 1
    B. 2
    C. 3
    D. 4 - 5
    E. 6 or more

28. The number of years that you have worked as a programmer, systems analyst, or software engineer:
    A. Never
    B. Less than one, part time
    C. One or more, part time
    D. Less than one, full time
    E. More than one, full time

29. The number of computer science courses that you finished in junior high or high school:
    A. None
    B. One semester
    C. One year
    D. Two years
    E. More than two years

30. Did you take the Advanced Placement (AP) Computer Science Course in high school?
    A. Yes
    B. No

31. How many hours of programming do you do for fun in a one week period, on the average?
    A. None
    B. One hour
    C. Two to five hours
    D. Five to fifteen hours
    E. Greater than fifteen hours

32 PROBLEM: A friend of yours has a dog who is overweight. The dog has been on a diet for several months. Your friend has been weighing his dog everyday.

Now your friend is curious about the range of weights that the dog has had. You agree to write a program that will sort the weights in ascending order and print a report.

The following code segment compiles the list and generates the report.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
                .
                .
                .
    for i := 1 to number_of_weights do
      begin
          wgt_index := 1;
          found := false;
          while (not found) and (wgt_index <= i) do
                if (weights [i] < update_weights [wgt_index])
                      then found := true
                else wgt_index := wgt_index + 1;
          for j := i downto wgt_index do
                update_weights [j + 1] := update_weights [j];
          update_weights [wgt_index] := weights [i]
      end;
    print_weights;

                .
                .
                .
```

Rate how interesting you found this problem/solution pair, on a scale from "A" to "E".

A - highly uninteresting
B .
C .
D .
E - extremely interesting

33. PROBLEM: You are asked by the government of a small country to develop an encoding algorithm for their "number of the day". Their number of the day is used as a password into their computer system in which all of the data about their extensive national park system is stored.

The following procedure produces a secret code.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
.
.
.
procedure secret_code (seed, key: integer; seed_table: seed_type;
                       var day_code: integer);
  var i: integer;
    begin
        day_code := truly_random(seed);
        for i := 1 to key do
            day_code := day_code *
                        seed_table(truly_random(i))
    end;
.
.
.
```

Rate how interesting you found this problem/solution pair, on a scale from "A" to "E".

A - highly uninteresting
B .
C .
D .
E - extremely interesting

34. PROBLEM: You are writing a program for the game, Jungle Escape. Jungle Escape is a board game for two players. In the game, the physical position of the player, in relation to the board, is a relevant parameter. In the computer version of the game, you must be able to rotate the board 90 degrees.

The following code segment accomplishes the required rotation.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
.
.
.
i := 1;
while (i <= boardsize) do
      begin
            j := 1;
            while (j <= boardsize) do
               begin
                  offset := (boardsize * (j - 1)) +
                                    ((boardsize - i) + 1);
                  new_offset := ((i - 1) * boardsize) + j;
                  new_board [new_offset] :=
                           board [offset];
                        j := j + 1
               end;
            i := i + 1
      end;
copyboard (board, new_board);
.
.
.
```

Rate how interesting you found this problem/solution pair, on a scale from "A" to "E".

A - highly uninteresting
B .
C .
D .
E - extremely interesting

35. PROBLEM: You are asked by your state government to develop an algorithm to generate a lottery number. This lottery number will be used as the winning number for the weekly Jackpot Bonanza.

The following procedure produces a lottery number.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
.
.
.
procedure number_generator (seed, key: integer;
                            var jackpot_number: integer);
    begin
 procedure number_generator (seed, key: integer;
                             var jackpot_number: integer);
     begin
          if (key = 1) then
               jackpot_number := random_time(seed)
          else
               jackpot_number := seed *
                    number_generator
                     (random_time(seed), (key -1),
                          jackpot_number)
     end;
.
```

Rate how interesting you found this problem/solution pair, on a scale from "A" to "E".

A - highly uninteresting
B .
C .
D .
E - extremely interesting

36. PROBLEM: You have just been elected as the treasurer of your favorite charitable organization. Your first job is to sort the contributions from a recent telephone campaign and print a report. The sorted contributions should be in order from generous to most generous.

The following code segment compiles the list and generates a report.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
        .
        .
        .
for i := 1 to number_of_contributions do
    for j := i to number_of_contributions do
        if (contribution [j] < contribution [i]) then
          begin
            save := contribution [j];
            contribution [j] := contribution [i];
            contribution [i] := save
          end;
print_report;
        .
        .
        .
```

Rate how interesting you found this problem/solution pair, on a scale from "A" to "E".

A - highly uninteresting
B .
C .
D .
E - extremely interesting

37. PROBLEM: A small jewelry store specializes in watch repairs. Not surprisingly, the store has a huge inventory of watch parts. Every day new parts are delivered. The owner and chief repair technician asks you to write an inventory program.

The following code segment updates the part catelogue.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
        .
        .
        .
read (delivery_item);
if (catelogue [1].part_number > 0) then
      begin
            updated := false;
            count := 2
      end
else updated := false;
while ((not updated) and (count <= catelogue [1].part_number))
    do
      if (item = catelogue [count].part_number) then
        begin
            catelogue [count].quantity :=
                  catelogue [count].quantity + 1;
                  updated := true
        end
      else count := count + 1

        .
        .
        .
```

Rate how interesting you found this problem/solution pair, on a scale from "A" to "E".

A - highly uninteresting
B .
C .
D .
E - extremely interesting

38. PROBLEM: You have just been appointed as the Official Magistrate of Development by the City Council. Your primary duty as Magistrate is to construct neighborhoods out of pre-fabricated houses. You must place houses on the street so that the front door faces the street.

This is a tedious job so you appoint a computer salesman as your assistant and receive a free computer as compensation.

You and your assistant write a program that will take a grid, which represents a pre-fabricated house, and reorient the grid on any street appropriately.

The following code segment accomplishes the required reorientation.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
.
.
.
for i := 1 to house_size do
   for j := 1 to house_size do
      new_house [(house_size * (j - 1)) +
            ((house_size - i) + 1)] :=
                  house [((i - 1) * house_size) + j];
display_house (house, new_house);
.
.
.
```

Rate how interesting you found this problem/solution pair, on a scale from "A" to "E".

A - highly uninteresting
B .
C .
D .
E - extremely interesting

39. PROBLEM: Your next exam in your computer science class consists of true and false problems. Your instructor is exhausted and offers to assign you an "A" in the class if you will write a test grading program.

The program will input the test answers for each student. You are to compare the student's answers with a key. The student must have at least 70% of the answers right to pass the exam. For each exam which is input, you are to report "Pass" or "Fail."

The following function finds a raw score for a single exam.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
.
.
.
function grade (first_question, last_question: integer; exam, key:
                exam_type) : integer;
  var score, i : integer;
    begin
        score := 0;
        for i := first_question to last_question do
            if exam [i] = key [i] then
                score := score +1;
        grade := score
    end;

.
.
.
```

Rate how interesting you found this problem/solution pair, on a scale from "A" to "E".

A = highly uninteresting
B .
C .
D .
E = extremely interesting

40. PROBLEM: You are to write a main procedure for an on-line ordering program. The program will be used by the Technocrat On-Line Shoe Company. Since Technocrat wants to sell as many pairs of shoes as possible, the buyer is allowed to use the ordering system as many times as he or she would like.

The following code segment is the main procedure for the ordering program.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
                    .
                    .
                    .
begin
      writeln ('enter y to order again,
                  anything else to quit');
      readln (answer);
      while (answer = 'y') do
        begin
          technocrat_profile;
          technocrat_match_by_fit;
          print_order_descript;
          writeln ('enter y to order again,
                       anything else to quit');
          readln (answer)
        end
end.
                    .
                    .
                    .
                    .
```

Rate how interesting you found this problem/solution pair, on a scale from "A" to "E".

A - highly uninteresting
B .
C .
D .
E - extremely interesting

41. PROBLEM: Northwest Ohio contains a wide variety of historical artifacts. An archeologist has a grant from a local historical society to collect and catelogue some of these valuable artifacts. This archeologist has organized the artifacts by type.

You are asked by the archeologist to write a program that maintains an inventory of the dig.

The following code segment updates the artifact catelogue.

Remember, the segment performs the operation correctly; any variables, functions, and procedures have been correctly declared elsewhere.

```
found := true;
done := false;
read (artifact-type);
while ((catelogue[0].number > 0) and (not done))
      begin
            first := 1;
            last := catelogue [0].number;
            found := false;
            done := true
      end
while ((last >= first) and (not found)) do
      begin
            middle := (first + last) div 2;
            if (artifact_type < catelogue [middle].number)
                  then last := middle - 1
            else if (artifact_type > catelogue [middle].number)
                  then first := middle + 1
            else begin
                  catelogue [middle].quantity :=
                        catelogue [middle].quantity + 1;
                  found := true
                end
      end
```

Rate how interesting you found this problem/solution pair, on a scale from "A" to "E".

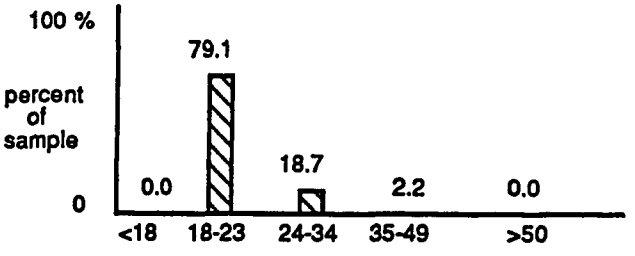
A - highly uninteresting
B .
C .
D .
E - extremely interesting

APPENDIX C

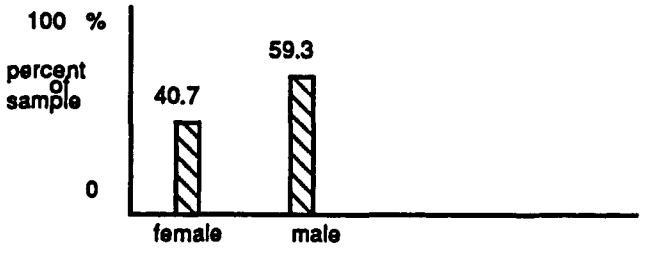
SAMPLE CHARACTERISTICS

(N=93)

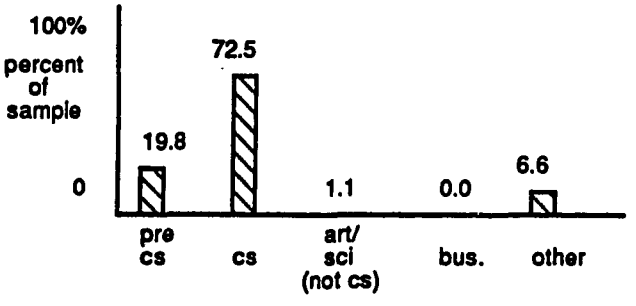
1. Age of subjects



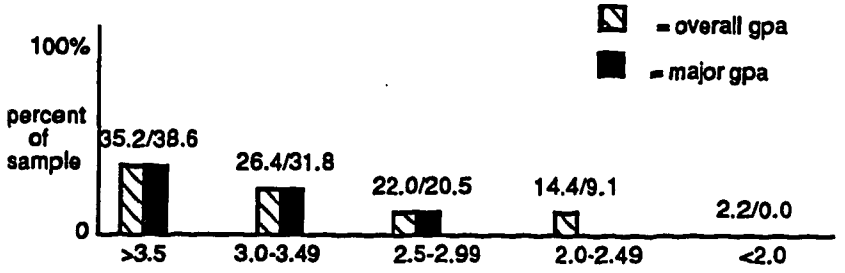
2. Gender of subjects (N=91)



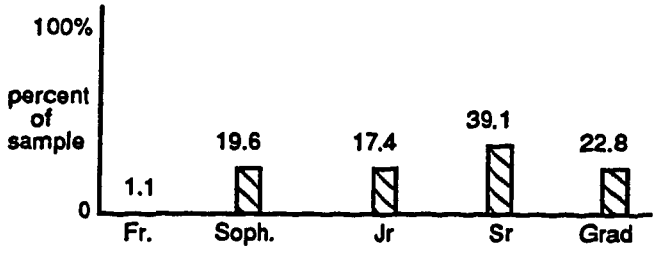
3. Majors of subjects (N=93)



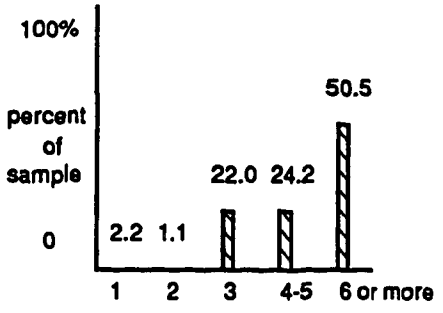
4. Grade point average (max = 4.0; N=93)



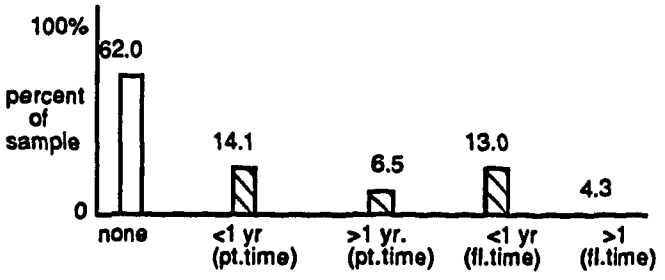
5. Rank of subjects (N=92)



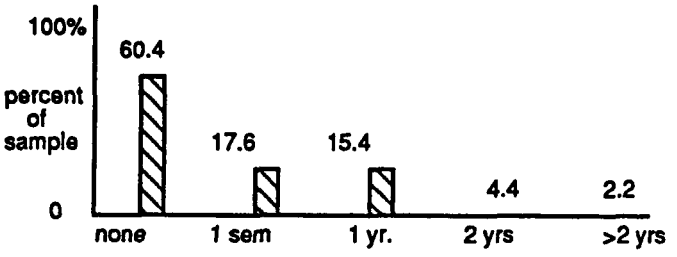
6. Number of college level computer science courses completed (N=93)



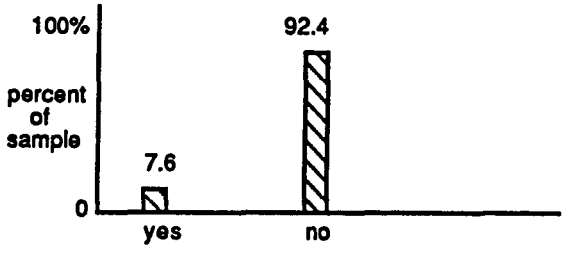
7. Related work experience of subjects (N=93)



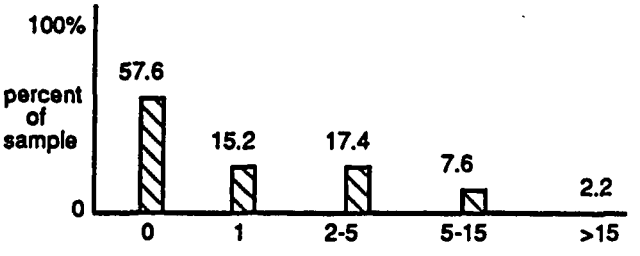
7. Number of computer science courses in high school (N=93)



8. Took Advanced Placement Computer Science Course (N=93)



9. Hours per week of programming for fun (N=93)

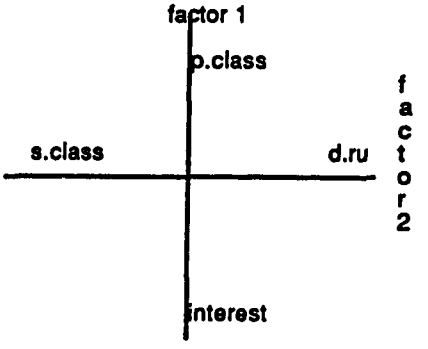


Appendix D - Derivation of Prediction Framework

Loadings:

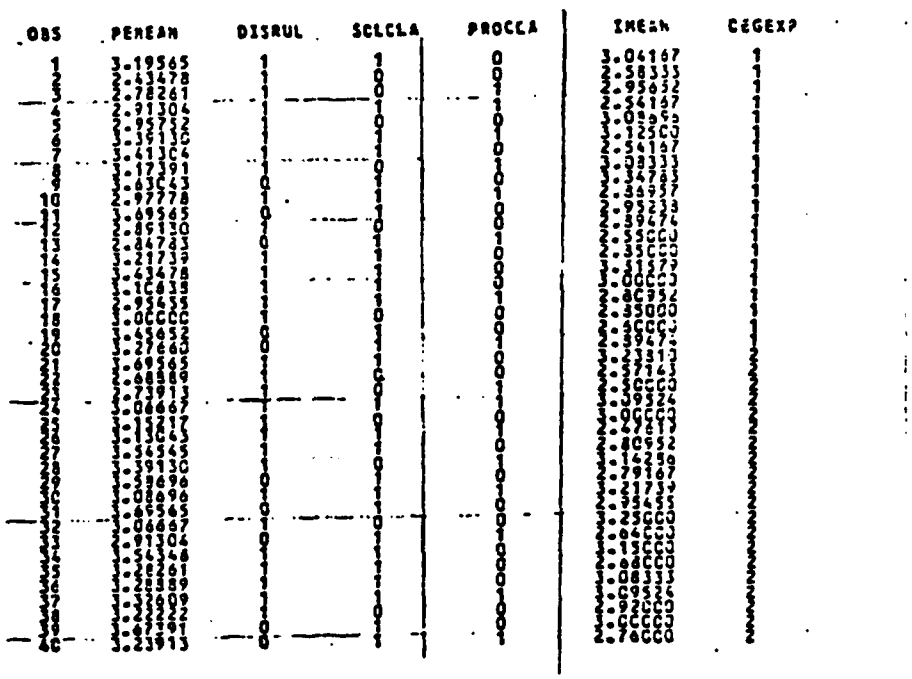
	Novices		Experts	
	Factor 1	Factor 2	Factor 1	Factor 2
interest	.04	-.74	-.07	.81
p. class.	-.01	.77	.00	.70
s. class	-.82	.08	.84	.10
dis. rule	.83	.02	-.82	.18

Facsimile of Plots:



APPENDIX E

FACSIMILE OF DATA USED IN ANALYSIS OF COVARIANCE



OBS	PEMEAN	DISRUL	SOLCLA	PROCLA	IMEAN	DEGEXP
1	3.19565	1	1	0	3.04167	1
2	2.43478	1	0	0	2.58333	1
3	2.78261	1	0	1	2.95833	1
4	2.91304	1	1	1	2.54167	1
5	2.95652	1	0	0	3.08333	1
6	3.26087	1	1	0	3.12500	1
7	3.41304	1	1	0	2.54167	1
8	3.17391	1	0	0	3.08333	1
9	3.43478	0	1	0	3.34783	1
10	2.97778	1	1	0	2.58697	1
11	2.69565	0	0	0	2.95238	1
12	2.86130	1	0	0	2.39474	1
13	2.84783	0	1	0	2.55000	1
14	3.21739	1	1	0	2.35000	1
15	3.43478	1	1	0	3.11579	1
16	3.10435	1	1	0	3.00000	1
17	2.95455	1	1	0	2.80952	1
18	3.00000	1	0	0	2.35000	1
19	3.45652	0	1	0	3.40000	1
20	3.27643	0	1	0	2.39474	1
21	3.69565	1	1	0	3.23810	1
22	3.68696	1	0	0	3.57143	1
23	3.73913	1	1	0	3.50000	1
24	3.08696	1	1	0	3.39524	1
25	3.15217	1	0	0	3.00000	2
26	3.13043	1	0	0	3.47619	2
27	3.54348	1	1	0	2.80952	2
28	3.39130	1	0	0	3.14286	2
29	3.58696	0	1	0	2.79167	2
30	3.08696	1	1	0	3.21739	2
31	3.69565	0	1	0	2.95455	2
32	3.06667	1	0	0	3.25000	2
33	2.91304	0	1	0	3.04762	2
34	3.54348	1	1	0	3.15000	2
35	3.26261	1	1	0	3.00000	2
36	3.28889	1	1	0	3.08333	2
37	3.32609	1	1	0	3.09524	2
38	3.23333	1	0	0	2.97500	2
39	3.67391	0	1	0	3.00000	2
40	3.23913	0	1	1	2.78000	2

where:

DISRUL = discourse rule violation status (0=no violations 1=1 or more viols.)

SOLCLA = solution classicalness (0=low class. 1=not low class.)

PROCLA = problem classicalness (0=not high class. 1=high class.)

IMEAN = mean-per-item interest

PEMEAN = mean-per-item preference

DEGEXP = expertise level (1=novice 2=expert)

Items:

1, 21 - Thunder Storm
2, 22 - Legislative Voting
3, 23 - Jewelry Store
4, 24 - Seawater
5, 25 - Lottery
6, 26 - Prices
7,27 - Quilting
8,28 - Toxic Substances
9, 29 - Test Grading
10, 30 - Archeology
11, 31 - Library System
12, 32 - Jungle Escape
13, 33 - Weights
14, 34 - South Pole
15, 35 - Shoe Store
16, 36 - Secret Code
17, 37 - Rare Flora
18, 38 - Arrange Houses
19, 39 - Pet Poison
20, 40 - Contributions

APPENDIX F

Analysis of Covariance for Two Subject Groups

Experts:

Model	F(15,4) = 6.70	p<.0027	
Model =	Discourse Rule	F = 3.95	p<.0654
	Interest	F = 11.57	p<.0040
	S. Class.	F = 4.87	p<.0434
	P. Class	F = 6.43	p<.0228

Novices:

Model	F(15,4) = 7.95	p<.0012	
Model =	Discourse Rule	F = 10.08	p<.0063
	Interest	F = 10.16	p<.0061
	S. Class.	F = 8.83	p<.0095
	P. Class	F = 2.73	p<.1195

APPENDIX G

EXPERT AND NOVICE FACTOR LOADINGS

DIMENSION

	1	2	3	4
1	0.3170	-0.4922	0.2176	-0.3506
2	0.0746	-0.0219	0.4696	-0.5242
3	-0.1326	0.1983	-0.0391	-0.3689
4	-0.3587	0.2680	-0.1731	-0.2120
5	0.0005	-0.0172	0.0411	-0.4311
6	-0.6326	-0.0366	0.1318	0.1323
7	-0.0627	0.0643	0.5581	0.0860
8	-0.4167	0.0930	0.1068	-0.0601
9	-0.2142	-0.2120	0.4224	-0.2012
10	0.0149	0.7206	0.0080	-0.3991
11	-0.0799	-0.1165	0.3440	-0.1068
12	-0.0694	0.2830	0.7351	0.0429
13	0.2697	0.0204	-0.1877	-0.3998
14	-0.1516	0.3458	0.3884	-0.0199
15	-0.3306	-0.1093	0.1539	-0.3539
16	-0.4005	0.1014	-0.1017	-0.4449
17	-0.0556	0.3495	0.0352	-0.0599
18	0.1120	-0.0991	0.2875	0.0615
19	-0.3399	-0.7587	0.0951	-0.3154
20	-0.7953	-0.0406	-0.0141	-0.0179

Expert Loadings

DIMENSION

	1	2	3	4	5
1	-0.2880	-0.1184	0.7176	0.0924	0.0658
2	-0.3061	-0.1118	-0.2844	-0.4842	-0.1184
3	-0.0792	0.0223	0.0626	-0.1534	-0.6175
4	-0.0677	0.0611	0.1962	-0.6090	-0.0841
5	-0.0976	-0.2107	0.0115	-0.6362	-0.0938
6	-0.0505	-0.6671	0.1484	-0.1789	0.0159
7	-0.2528	0.0111	-0.1216	-0.1297	-0.1252
8	-0.3884	0.0814	0.2262	0.0377	-0.2934
9	-0.4637	-0.1161	0.3511	-0.2863	0.1318
10	-0.0001	-0.0448	-0.1923	-0.4628	-0.0725
11	-0.0672	0.0833	0.2490	0.2167	-0.2167
12	-0.5445	0.1242	0.0921	0.0174	0.1494
13	-0.5426	-0.2028	-0.1030	-0.1469	-0.1218
14	0.0580	-0.0527	0.0023	-0.0681	-0.7741
15	-0.3959	-0.1655	0.0843	0.0692	-0.0370
16	0.0027	-0.2002	0.3228	-0.2683	-0.3795
17	-0.5146	0.0289	0.0533	-0.0763	0.0564
18	-0.5230	-0.2808	-0.2036	-0.1370	-0.3174
19	0.1523	-0.1237	0.7427	0.0161	-0.2314
20	-0.1020	-0.9590	0.0387	-0.0394	-0.0461
ROOT=	1.96	1.69	1.68	1.57	1.53

Novice Loadings

BIBLIOGRAPHY

BIBLIOGRAPHY

Adelson, B. (1984). When Novices Surpass Experts: The Difficulty of a task May Increase with Expertise. Journal of Experimental Psychology: Learning, Memory, and Cognition, 10, 483-495.

Arblaster, A.T. (1983). The Evaluation of a Programming Support Environment. in T.R.G. Green, S.J. Payne, & van der Veer, G.C. (Eds.), The Psychology of Computer Use. (pp. 191-221). New York: Academic Press.

Bentley, J. (1986). Literate Programming. Communications of the ACM, 29, 364-369.

Black, J.B. (1984). Understanding and Remembering Stories. in J.R. Anderson, J.R. and S.M Kosslyn (Eds.), Tutorials in Learning and Memory: Essays in Honor of Gordon Bower. (pp. 235-255). San Francisco: W.H. Freeman and Company.

Black, J.B., & Bern, H. (1981). Causal Coherence and Memory for Events in Narratives. Journal of Verbal Learning and Verbal Behavior, 20, 267-275.

Black, J.B., Galambos, J.A., & Read, S.J. (1984). Comprehending Stories and Social Situations. in R.S. Wyer and T.K. Srull (Eds.), Handbook of Social Cognition. 3 (pp. 45-86). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Blalock, H.M. (1979). Social Statistics, 2nd Edition. New York: McGraw-Hill, Inc.

Brooks, R.E. (1980). Studying Programmer Behavior Experimentally: The Problems of Proper Methodology. Communications of the ACM, 23, 207-213.

Chi, M.T.H., Feltovich, P.J., & Glaser, R. (1981). Categorization and Representation of Physics Problems by Experts and Novices. Cognitive Science, 5, 121-125.

Child, D. (1973). The Essentials of Factor Analysis. New York: Holt, Rinehart and Winston.

Child, I.L. (1981). Bases of Transcultural Agreement in Response to Art. in H.I Day (Ed.), Advances in Intrinsic Motivation and Aesthetics. (pp. 415-432). New York: Plenum Press.

Cioch, F.A. (1985). Software Understandability: An Empirical Study. Unpublished doctoral dissertation, The University of Michigan, Ann Arbor, MI.

Crozier, J.B. (1981). Information Theory and Melodic Perception: In Search of the Aesthetic Engram. in H.I. Day (Ed.), Advances in Intrinsic Motivation and Aesthetics. (pp. 433- 461). New York: Plenum Press.

Dale, N., & Orshalick, D. (1983). Introduction to PASCAL and Structured Design. Lexington, MA: D.C. Heath and Company.

Daniel, T.C., & Vining, J. (1983). Methodological Issues in Assessment of Landscape Quality. in I. Altman and J.F. Wohlwill (Eds.), Behavior and the Natural Environment. (pp. 39-84). New York: Plenum.

Dzida, W., Herda, S., & Itzfeldt, W.D. (1978). User-Perceived Quality of Interactive Systems. IEEE Transactions on Software Engineering, SE-4, 270-276.

Egan, D.E., & Schwartz, B.J. (1979). Chunking in Recall of Symbolic Drawings. Memory and Cognition, 7, 149-158.

Gelman, E., Rogers, M., Lubenow, G.C., Marbach, W.D., Friday, C., & Cook, W.J. (1985). Showdown in Silicon Valley. Newsweek, CVI, 46-50.

Gilfoil, D.M. (1982). Warming Up to Computers: A Study of Cognitive and Affective Interaction Over Time. in Proceedings Human Factors in Comptuer Systems. Gaithersburg, MD. (pp. 245-253).

Hagglund, S.L., & Tibell, R. Multi-Style Dialogues and Control Independence in Interactive Software. in T.R.G. Green, S.J. Payne, and G.C. van der Veer (Eds.), The Psychology of Computer Use. (pp. 171-189). New York: Academic Press.

Hamming, R.W. (1980). The Unreasonable Effectiveness of Mathematics. The American Mathematical Monthly, 87, 81-90.

Hare, F.G. (1981). Recent Developments in Experimental Aesthetics: A Summary of Berlyne Laboratory Research Activities, 1974-1977. in H.I. Day (Ed.), Advances in Intrinsic Motivation and Aesthetics. (pp. 487-500). New York: Plenum Press.

Harman, H.H. (1976). Modern Factor Analysis, 3rd ed. Chicago: University of Chicago Press.

Hayes, J.R., & Simon, H.A. (1976). The Understanding Process: Problem Isomorphs. Cognitive Psychology, 8, 165-190.

Herzog, T.R., & Larwin, D.A. (unpubl.). The Appreciation of Humor in Captioned Cartoons.

Huttenlocher, J. (1976). Language and Intellegence. in L.B. Resnick (Ed.), The Nature of Intellegence. Hillsdale, NJ: Erlbaum.

Jeffries, R., Turner, A.A., Polson, P.G., Atwood, M.E. (1981). The Processes Involved in Designing Software. in J.R. Anderson (Ed.), Cognitive Skills and their Acquisition. (pp. 255-283). Hillsdale, NJ: Lawrence Erlbaum Associates.

Kahney, H. (1983). What Do Novice Programmers Know about Recursion. in Proceedings CHI"83 Human Factors in Computer Systems. Boston, MA. (pp. 235-239).

Kaplan, R. (1975). A Strategy for Dimensional Analysis. in D.H. Carson (Ed.), Man-Environmental Interactions: Evaluations and Applications. (pp. 66-68). : Dowden, Hutchinson & Ross.

Kaplan, S. (1987). Aesthetics, Affect, and Cognition: Environmental Preference from an Evolutionary Perspective. Environment and Behavior, 19.

Kaplan, S. (1977). Participation in the design process: A cognitive approach. in D. Stokals (Ed.), Perspective on environment and behavior. (pp. ch. 10). New York: Plenum.

Kaplan, S. (1978). Perception of an uncertain environment. in S. Kaplan and R. Kaplan ((Eds.).), Humanscape: Environments for People. Belmont, CA: Duxbury.

Kaplan, S., & Kaplan, R. (1982). Cognition and Environment. New York: Praeger.

Kernighan, B., & Plauger, P. (1978). The Elements of Style. New York: McGraw Hill Co.

Larkin, J.H. (1983). The Role of Problem Representation in Physics. in D. Gentner and A.L. Stevens. (Eds.), Mental Models. (pp. 75-98). Hillsdale, NJ: Lawrence Erlbaum Associates.

Leventhal, L.M., & Mynatt, B.T. (in press). Components of Typical Undergraduate Courses in Software Engineering: Results from a Survey. IEEE Transactions on Software Engineering.

Lewis, C. (1981). Skill in Algebra. in J. R. Anderson (Ed.), Cognitive Skills and Their Acquisition. (pp. 85-110). Hillsdale, NJ: Erlbaum.

Lingoes, J.C. (1972). A General Survey of the Guttman-Lingoes Nonmetric Program Series. in R.N. Shepard, A.K. Romney, and S.B. Nerlove (Eds.), Multidimensional Scaling. 1 New York: Seminar.

Luria, A.R. (1973). The Working Brain. : Penguin.

Maass, S. (1983). Why Systems Transparency. in T.R.G. Green, S.J. Payne,and G.C. van der Veer (Eds.), The Psychology of Computer Use. (pp. 19-28). New York: Academic Press.

Malone, T.W. (1981). Toward a Theory of Intrinsically Motivating Instruction. Cognitive Science, 4, 333-369.

McKeithen, K.B., Reitman, J.S., Reuter, H.H., & Hirtle, S.C. (1981). Knowledge Organization and Skill Differences in Computer Programmers. Cognitive Psychology, 13, 307-325.

Moher, T., & Schneider, G.M. (1982). Methodology and Experimental Research in Software Engineering. International Journal of Man-Machine Studies, 16, 65-87.

Moher, T., & Schneider, G.M. (1981). Methods for Improving Controlled Experimentation in Software Engineering. in Proceedings of the 5th International Conference on Software Engineering. (pp. 224-233). : IEEE.

Molzberger, P. (1983). Aesthetics and Programming. in Proceedings CHI'83, Human Factors in Computing Systems. Boston, MA. (pp. 247-250). New York: ACM.

Molzberger, P. (1984). Transcending the Basic Paradigm of Software Engineering. (Bericht Nr. 8405). Neubiberg: Hochschule der Bundeswehr Monchen .

Moran, T. (1981). Guest Editor's Introduction ... An Applied Psychology of the User. Computing Surveys, 13, 1-11.

Moynihan, C., & Mehrabian, A. (1981). The Psychological Aesthetics of Narrative Forms. in H.I. Day (Ed.), Advances in Intrinsic Motivation and Aesthetics. (pp. 323-340). New York: Plenum Press.

Mynatt, B. T. (1984). The effect of semantic complexity on the comprehension of program modules. International Journal of Man-Machine Studies, 21, 91-103.

Nelson, D.L., & Castano, D. (1984). Mental Representations for Pictures and Words: Same or Different. American Journal of Psychology, 97, 1-15.

Newell, A., & Simon, H.A. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice-Hall.

Nicki, R.M. (1981). Ambiguity, Complexity, and Preference for Works of Art. in H.I. Day (Ed.), Advances in Intrinsic Motivation and Aesthetics. (pp. 365-383). New York: Plenum Press.

Rushinek, A., & Rushinek, S.F. (1986). What Makes Users Happy. Communications of the ACM, 29, 594-598.

Savitch, W.J. (1984). PASCAL: An Introduction to the Art and Science of Programming. Menlo Park, CA: The Benjamin/Cummings Publishing Co., Inc.

Sayward, F.G. (1984). Experimental Design Methodolgies in Software Science. Information Process and Management, 20, 223-227.

Sengler, H.E. (1983). A Model of Understandability of a Program and Its Impact on the Design of the Programming Language GRADE. in T.R.G. Green, S.J. Payne,and G.C. van der Veer (Eds.), The Psychology of Computer Use. (pp. 91-106). New York: Academic Press.

Sheil, B.A. (1981). The Psychological Study of Programming. Computing Surveys, 13, 101-120.

Shneiderman, B. (1976). Exploratory Experiments in Programmer Behavior. International Journal of Computer and Information Sciences, 5, 123-143.

Shneiderman, B. (1979). Human Factors Experiments in Designing Interactive Systems. Computer, 9-19.

Soloway, E., Ehrlich, K., & Black, J.B. (1983). Beyond Numbers: Don't Ask "How Many". . . Ask "Why". in Proceedings CHI'83 Human Factors in Computer Systems. Boston. (pp. 240-246).

Soloway, E., Ehrlich, K., & Bonar, J. (1982). Tapping into Tacit Programming Knowledge. in Proceedings Human Factors in Computing Systems. Gaithersburg, MD. (pp. 52-58).

Soloway, E., & Ehrlich, K. (1984). Empirical Studies of Programming Knowledge. IEEE Transactions on Software Engineering, SE-10, 595-609.

Spohrer, J.C., Pope, E., Lipman, M., Sack, W., Freiman, S., /Littman, D., Johnson, L., & Soloway, E. (1985). Bug Catalogue II, III, IV. (YALEU/CSD/RR #386). New Haven, CT.

Voss, J.F., Tyler, S.W., & Yengo, L.A. (1983). Individual Differences in the Solving of Social Science Problems. in R.F. Dillon, & R.R. Schmeck (Eds.), Individual Differences in Cognition. 1 (pp. 205-232). : Academic Press.

Walker, E.L. (1981). The Quest for the Inverted U. in H.I. Day (Ed.), Advances in Intrinsic Motivation and Aesthetics. (pp. 39-70). New York: Plenum Press.

Weiser, M.D. (1979). Program Slices: Formal, Psychological, and Practical Investigations of an Automatic Program Abstraction Method. Unpublished doctoral dissertation, The University of Michigan, Ann Arbor, MI.

Weiser, M., & Shertz, J. (1983). Programming Problem Representation in Novice and Expert Programmers. International Journal of Man-Machine Studies, 19, 391-398.

Wildt, A.R., & Ahtola, O.T. (1978). Analysis of Covariance. (E.M. Uslaner, Ed.) (Paper Series on Quantitative Applications in the Social Sciences Series number 07-012). Beverly Hills, CA: Sage Publications.

Zajonc, R.B. (1980). Feeling and Thinking: Preferences Need No Inferences. American Psychologist, 35, 151-175.