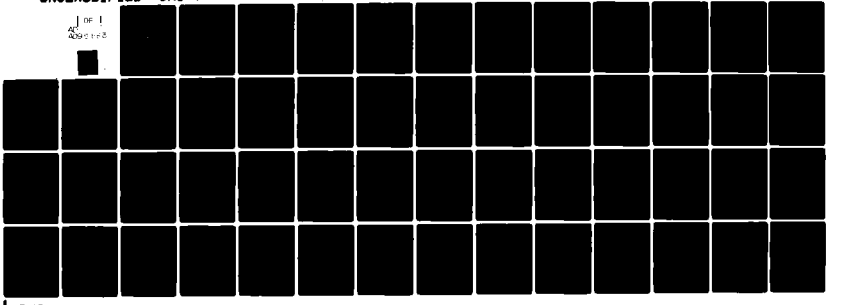
AD-A095 863 SCIENCE APPLICATIONS INC ENGLEWOOD CO F/G 9/2
STUDIES IN PLAN CONSTRUCTION. II. NOVICE DESIGN BEHAVIOR.(U)
DEC 80 M E ATWOOD, R JEFFRIES N00014-78-C-0165
UNCLASSIFIED SAI-80-154-DEN NL



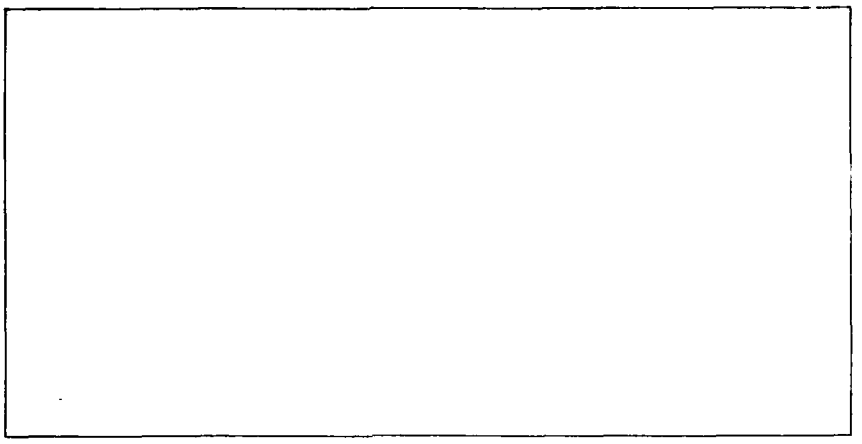
END
DATE FILMED
4-81
DTIC

LEVEL II

12

SR

AD A095863



SCIENCE APPLICATIONS INCORPORATED

Studies in Plan Construction II: Novice Design Behavior

Technical Report

SAI-80-154-DEN

December 1980

Michael E. Atwood

Science Applications, Inc.

Robin Jeffries

Carnegie-Mellon University

This research was sponsored by the Personnel and Training Research Programs, Psychological Sciences Division, Office of Naval Research, under Contract No. N00014-78-C-0165, Contract Authority Identification Number NR157-414. Computer time was provided by the SUMEX-AIM Computing Facility at the Stanford University School of Medicine which is supported by grant RR-00785 from the National Institutes of Health. Order of authorship is alphabetical to reflect equal contributions of both authors.

Approved for public release; distribution unlimited.

Science Applications, Inc.

40 Denver Technological Center West, 7935 East Prentice Avenue, Englewood, Colorado 80111, 303/773-6900

Other SAI Offices: Albuquerque, Ann Arbor, Arlington, Atlanta, Boston, Chicago, Huntsville, La Jolla, Los Angeles, McLean, Palo Alto, Santa Barbara, Sunnyvale, and Tucson.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A095 863
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) Studies in Plan Construction. II. Novice Design Behavior	5. TYPE OF REPORT & PERIOD COVERED Technical Report,
	6. PERFORMING ORG. REPORT NUMBER SAI-80-154-DEN
7. AUTHOR(s) Michael E. Atwood and Robin Jeffries	8. CONTRACT OR GRANT NUMBER(s) N00014-78-C-0165, PHS-RR-00775
9. PERFORMING ORGANIZATION NAME AND ADDRESS Science Applications, Inc. 7935 E. Prentice Avenue Englewood, CO 80111	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61153N RR 042-06 RR 042-06-02 NR157-414
11. CONTROLLING OFFICE NAME AND ADDRESS Personnel & Training Research Programs Office of Naval Research Arlington, VA 22217	12. REPORT DATE December 1980
	13. NUMBER OF PAGES 38
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Software design is the process of translating a set of task requirements into a structural description of a computer program that will perform the indicated task. Expert designers rely on a great deal of domain-specific knowledge to perform a design task. Novice designers lack this specialized knowledge and rely on more general problem solving strategies. This report describes the novices' design processes and illustrates their operation by considering thinking-aloud protocols.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Table of Contents

List of Figures

Abstract

Software design is the process of translating a set of task requirements into a structural description of a computer program that will perform the indicated task. Expert designers rely on a great deal of domain-specific knowledge to perform a design task. Novice designers lack this specialized knowledge and rely on more general problem solving strategies. This report describes the novices' design processes and illustrates their operation by considering thinking-aloud protocols.

Introduction

In an earlier report (Atwood and Jeffries, 1980), we described the problem solving processes used by skilled software designers. We then considered the nature of the high-level knowledge structures that guide expert behavior in a design task (Jeffries, Turner, Polson, and Atwood, 1981). In this report, we focus on the behavior of less skilled designers.

Solving a design problem involves the identification of design components followed by the solution of these components. The identified components represent subproblems to be solved and their solution involves the creation and satisfaction of goals. The form and content of these goal structures and the manner in which they are generated, therefore, provides a record of the problem solving actions taken by a designer. In this report, as in the earlier study of experts' behavior, we will examine these goal structures and the problem solving processes involved in their creation. The nature of the more global problem solving processes underlying novice design behavior will be ascertained from these local descriptions.

The expert designer's knowledge is very domain specific. Experts know a lot about their area of expertise; their behavior is strongly driven by this domain-specific knowledge. We refer to the problem solving processes that guide expert behavior as a *control schema*, a specialized collection of knowledge and procedures. For software design, we call this control schema the *design schema*. Novices, on the other hand, don't have this rich store of domain-specific knowledge. As a result, they resort to the use of more general problem solving strategies. We consider here the nature of these general processes and the manner in which they are manifested in a design task.

Task and Problem Description

Software design can be viewed as the process of translating a set of task requirements (functional specifications) into a structural description of a computer system that will perform the indicated task. There are three major elements of this description. First, the specifications are decomposed into a collection of modules, each of which satisfies part of the problem requirements. Second, the designer must specify the relationships and interactions

among the modules. This includes control structures, which indicate the order in which modules are activated and the conditions under which they are used. Third, a design includes a definition of the data structures to be used.

The particular problem to be discussed in this paper is to design a page-keyed indexing system. This problem is assumed to be of moderate difficulty, in that a reasonable design could be constructed using only the techniques taught in upper-division undergraduate computer science courses. Figure 1 shows the problem description given to the subjects.This is a slightly simplified version of the problem given to our first two expert subjects (S2 and S3; see Atwood and Jeffries, 1980). We rewrote the specifications to eliminate some of the more problematic aspects (e.g., how to deal with hyphenated words) in an effort to shorten the time needed to solve the problem.

Thinking out loud protocols were collected from five undergraduate students enrolled in an assembly language programming course. They were instructed to construct a design for the page-keyed indexing problem and then to summarize their design and to describe the techniques and procedures used to generate a solution. Two protocols were selected for detailed analysis (S17 and S19). Both subjects had completed a course that specifically taught software design concepts, although this was not a factor in their selection. In comparison with our expert subjects, who had several years of actual design experience, we consider these subjects to be novices.

Method of Analysis

We analyzed the protocols by recoding them into a set of condition-action pairs (for a more detailed discussion, see Atwood and Jeffries, 1980). Each rule is intended to represent a decision made by the subject together with the features of the problem situation that motivated that decision. We stayed as close as possible to the protocol itself, but motivating conditions and occasionally entire rules were inferred as necessary to keep the set of rules reasonably complete and consistent.

Each rule is written in an IF...THEN format. We permit three categories of information to occur in the rules: policies, goals, and notes. Policies are general strategies that a particular

PAGE-KEYED INDEXING SYSTEM

Background.

A book publisher requires a system to produce a page-keyed index. This system will accept as input the source text of a book and produce as output a list of specified index terms and the page numbers on which each index term appears. This system is to operate in a batch mode.

Design Task.

You are to design a system to produce a page-keyed index. The source file for each book to be indexed is an ASCII file residing on disk. Page numbers will be indicated in the form *NNNN where "*" is a marker character used to identify the occurrence of page numbers and NNNN is the page number. "*" is a "reserved" character and will not appear anywhere else in the text.

The page number will appear after a block of text that comprises the body of the page. The page numbers will be in ascending order, but not necessarily in sequential order. A book may contain pages that consist of illustrations or figures. Since such pages are not to be indexed, they are not included in the source file. Normally, a page contains enough information to fill an 8 1/2 X 11 inch page. Each page of text is stored as a single record. Each word is preceded by a single blank and may be followed by a single punctuation mark. In addition, single words do not cross page boundaries and there are no hyphenated words.

A term file, containing a list of terms to be indexed, will be read from a card reader. The term file contains one term per line, where a term is 1 to 5 words long. The term file will be input in alphabetical order. All terms start in column 1 of the card and words are separated by single blanks.

The system should read the source file and term file and find all occurrences of each term to be indexed. The output should contain the index terms listed alphabetically with the page numbers following each term in numerical order.

Figure 1: Page-Keyed Indexer Problem

designer uses in all or most design tasks. Decisions such as *the efficiency of the program is important* or *data structures must be designed first* would be represented as policies. Goals represent the tasks the designer proposes to accomplish in order to solve a particular problem; a decision to *construct the data structure for the terms* or to *determine what to do when end of file is reached* would be encoded as goals. The final category is notes, which are facts that are entered into or retrieved from long term, working, or external memory. Any information that the designer generates or notices is included as notes. Examples are that *the terms should be stored as an array* or that *a binary tree enables one to efficiently search a list of items.*

The left hand side of each rule contains one or more clauses that are true of the current situation. These can include active goals and policies or notes retrieved from long-term, working or external memory. The action clauses on the right hand side of each rule lead to the generation of new information. They either activate policies, create goals, or note information in the appropriate memory. Thus, each rule attempts to capture a single design-related decision and the information that we perceived the subject used in the decision making process.

Because of their narrow focus on individual decisions, the rules encode only the local aspects of the problem solving process. More global properties of subjects' design behavior must be inferred from these translations. In this report, we examine in detail the local and then the global characteristics of the design behavior of two novice designers.

Overview of the Protocols

Summaries of the protocols of S17 and S19 are given below. Each paragraph includes line numbers referencing the corresponding segment of the protocol for the reader who desires to examine parts of the protocol itself (available from the authors). The summaries are intended to provide a general context against which to consider the analyses of the following sections.

S17

S17's protocol consists of 527 lines, of which the first 265 were analyzed in detail. It is in this section of the protocol that S17 generates a design. In the remainder of the protocol, the subject describes the design to the experimenter and produces a summary.

S17 makes a single iteration over the problem, decomposing it into a modular structure with associated control constructs (IF/THENs, WHILEs). He then makes two additional passes over the design, at the urging of the experimenter to provide more detail. During these passes, S17 notices new information and adds steps to the design, but makes no attempt to decompose any of the modules into submodules. The design produced as a result of the initial decomposition and subsequent passes is shown in Figure 2.

S17 begins by reading the specifications, noting things that presumably will affect the form of the design (31-60). He comments on such things as the fact that the text file is an ASCII file, that the asterisk character has special status, and that pages will be in sequential order but not consecutive order. The last causes him some confusion, which is cleared up by the experimenter.

Next, S17 considers the sources of the two input files (61-67). Reviewing the specifications, he notes that the source text is on a file and the terms to be indexed will be read from a card reader. He elects to input the terms and store them in a vector (68-81), although the form of the vector is not considered at this time.

S17 then decides to search the source file for occurrences of each of the terms on a page by page basis (82-85). He notes that page boundaries can be identified by the asterisk-page number delimiter. The page being processed is to be stored as a binary tree (86-88).

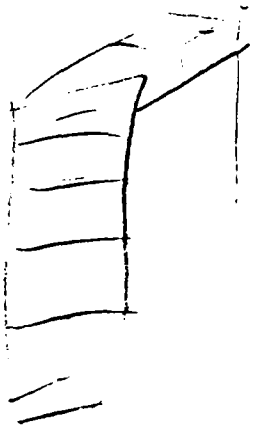
S17 then identifies a module to SEARCH SOURCE FOR TERMS (89-108). This module will iterate over the terms; if a term is found, the page number is to be recorded. He next decides to store the page number in the data structure that holds the terms (109-131). This causes him to change his vector for storing the terms into a matrix, in which the "first column is the vector of terms and columns 2 to N are pages".

S17 next considers the possibility that a term is not found on the page being processed (132-136). He notes that this merely implies that the iteration over the terms continues and

IN to be indexed

Source

~~File~~



Indexs

on

Term File

Executive

Get Terms to Index

(~~Vector~~ Matrix) 1st col Vect. Terms

2- n col pages

Get Page From Source

— Read to *NNNN

Save Page # (~~Header~~

~~Store Page in Binary Ice~~

Form

Search Source for Index Te

(00- To End of ~~File~~ FRANK

Find Term

Note Page in Term

Inner Matr

2 Not found on Page Loop

Iterate next ...g

Figure 2: Subject S17 -- Design

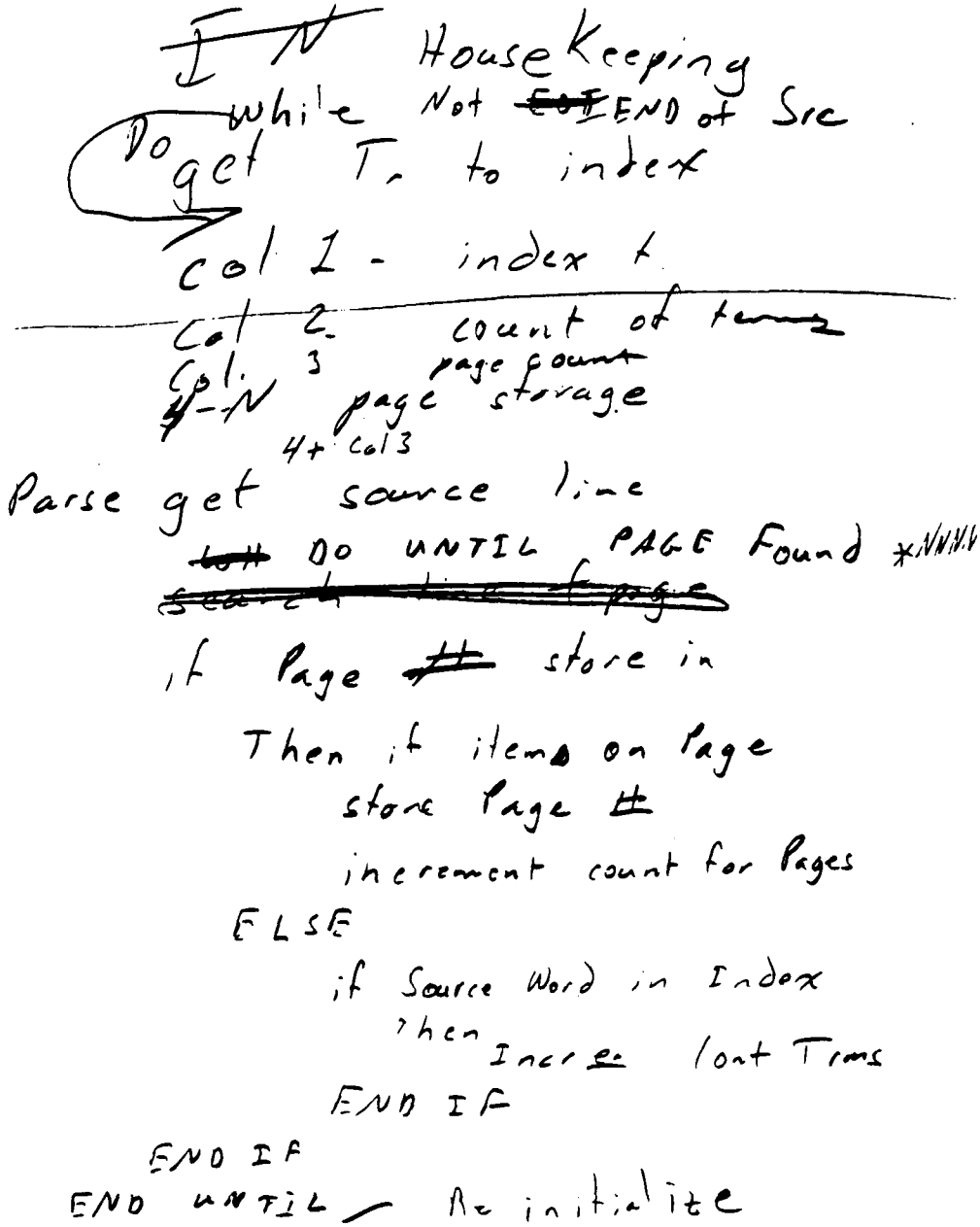


Figure 2, continued

no other processing is needed. This pass at a design is now complete, and he asks the experimenter "how much more broken down do you want this?".

At the request of the experimenter, S17 describes the design (136-170). This leads him to consider two changes. First, he mentions adding another column to his term matrix to accumulate the number of times each term is referenced and to serve as an index to the next free entry in that row. Second, he discusses two ways to interface the routine that reads the text to the comparison routine. One is to read a page of text, store it as a binary tree, and use that structure to compare to the terms. The other is to read the text a word at a time, comparing each word to the term list immediately, without storing it. S17 prefers to leave this decision to the implementer of the program.

It is clear at this point that S17 considers the design to be complete, but at the urging of the experimenter, he goes on to describe "lower modules". He attempts to expand the GET TERMS TO INDEX module (171-182) by noting that this module will insert terms in the data structure defined earlier and that it will terminate when the end of the term file is reached. Reexamination of the GET PAGE FROM SOURCE module (183-184) brings up again the decision of whether or not to store the text a line or word at a time to avoid storing the entire page. Consideration of the SEARCH SOURCE FOR TERMS module (184-186) only results in a reiteration of the earlier decision to record the page number if a match is found.

At this point the experimenter asks "where's the page number?" S17 realizes that he will not know the page number at the point that a match occurs, and this leads him to reconsider in detail his decision not to read and store an entire page of text (187-226). He decides that not storing the page is more storage efficient, and that it can be accomplished by keeping a "count" of whether or not a particular term has occurred on the current page.

S17 again considers the design to be complete, but attempts to provide more detail at the request of the experimenter. An attempt is made to expand the GET TERMS TO INDEX module by describing the data structure in which the terms and page numbers will be stored (227-237). The structure is defined to include the term, the "count" just mentioned, and the references themselves, but not the index to the current reference that he mentioned earlier.

S17 next considers the GET PAGE FROM SOURCE module (238-242). It is decided to read

the text a line at a time and to break each line into words. In reconsidering what must be done when the page number is found, S17 recalls his previous addition to the data structure and reincorporates it into the array (242-253). Finally, the details of the iteration over the text file are described (254-265).

In the remainder of the protocol (266-527), S17 again summarizes the design for the experimenter. This segment is primarily a review, but two issues are brought up that modify the design. First, up to this point, S17 has apparently been assuming that terms are single words. While re-reading the specifications (329-350), he notes that a term can be from one to five words long. He decides to pass this information on to the implementing programmer, as it might affect the form of the data structures.

Second, in response to a query from the experimenter about how the compare operation would work, S17 sketches out in a few sentences some aspects of an algorithm to compare multi-word terms (354-381). This leads him to notice that terms could cross page boundaries (382-417), and at the experimenter's insistence he outlines a method for dealing with such terms.

S19

S19's protocol consists of 711 lines, of which the first 447 were analyzed in detail. In the remainder of the protocol, the subject first reviews the design and then summarizes it. During the review, several modifications and corrections are made, but no further decomposition is attempted. Such episodes were also analyzed. The final design produced by S19 is shown in Figure 3. It was elaborated through two major passes. In the first, a modular decomposition was identified; in the second, the modules were described in more detail, but only in a few cases was further decomposition attempted.

While reading the problem specifications (50-84), S19 comments on such things as the sources of the input files, that the page numbers will be in numeric order, and that the text will contain both upper and lower case letters. Next, the major components of the design are enumerated (85-106). After both input files have been accessed, the next term will be selected, and the entire source file will be searched for that term. On a match "some kind of

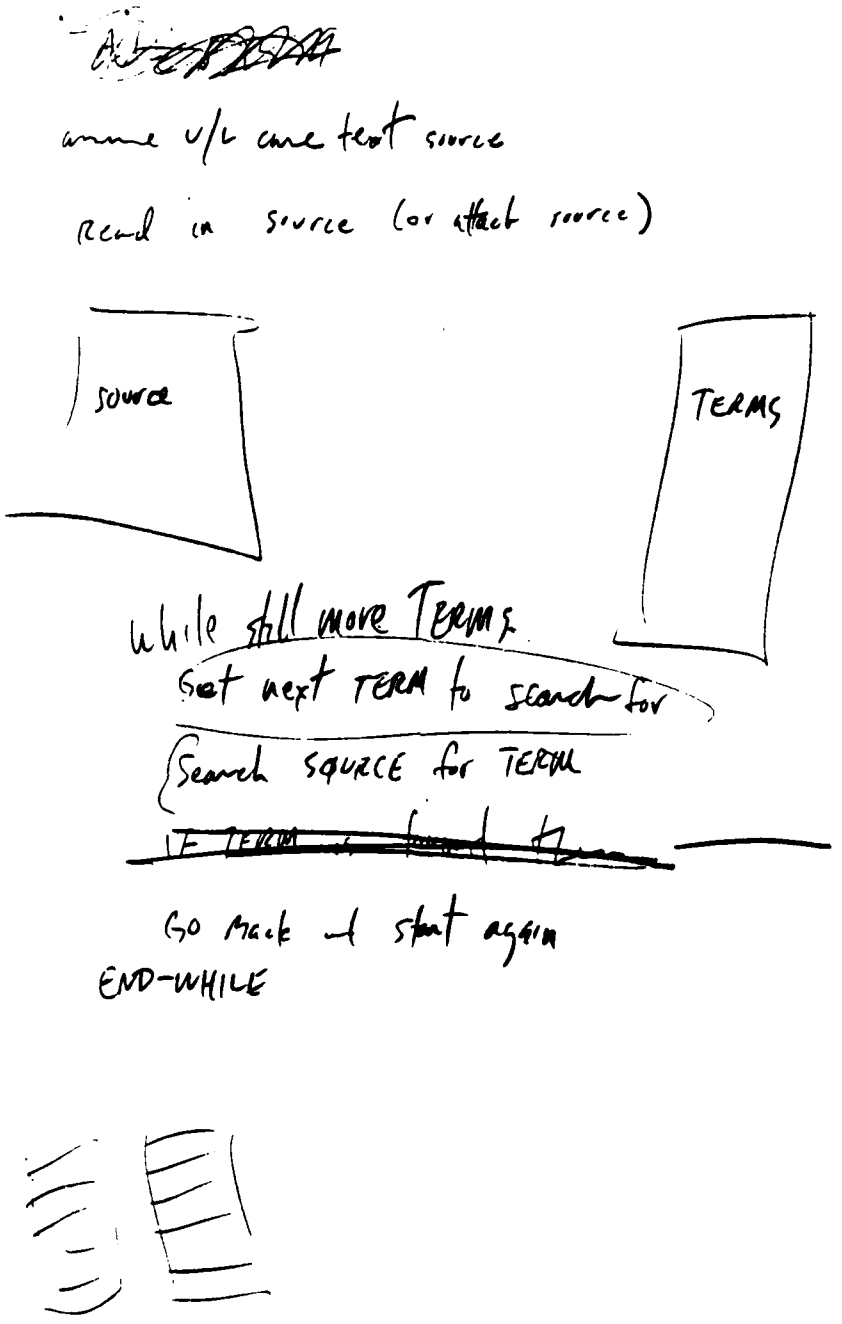


Figure 3: Subject S19 -- Design

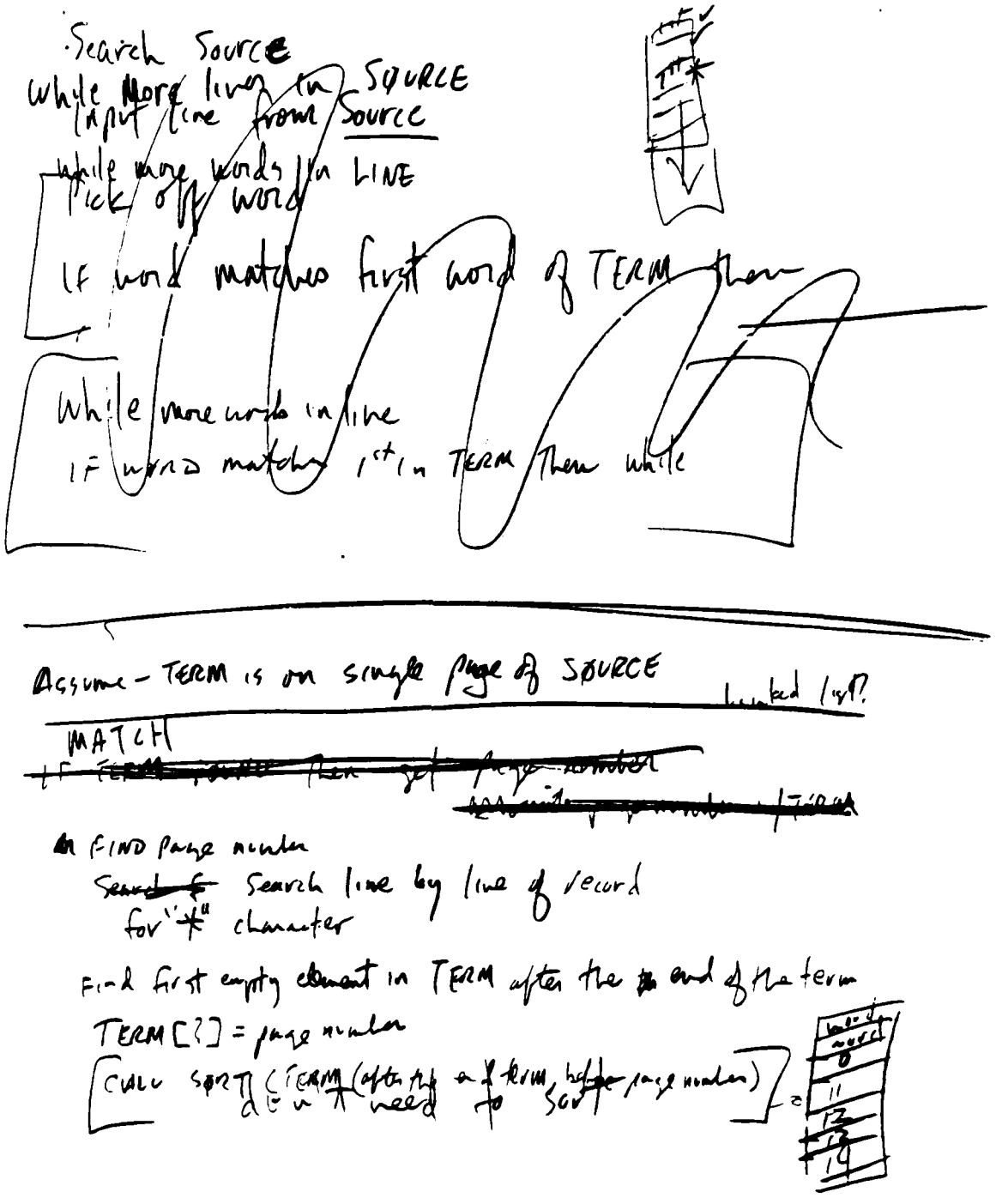


Figure 3, continued

Assume 100 page max on source

ATTACH SOURCE and TERMS
GET-NEXT-TERM
READ IN ONE TERM into ARRAY (105) 1-5 TERM 6-10 5 pages
IF END-OF-TERMS - EXIT (cleanup)
Assume TERM cannot cross over period.

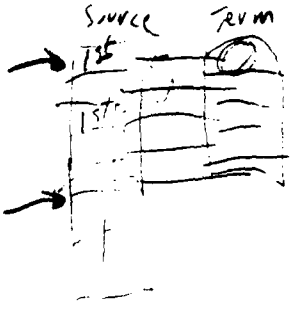
Source SOURCE for TRIM (01

READ IN one word. of Source
STRIP punctuation except period.

WHILE
IF WORD from source ≠ 1st word in term AND NOT EOF.
READ IN next word from source
END-WHILE
IF EOF Go back & start w/another TERM

IF next word from source ≠ 2nd in Term (if not True) Go to M

EXTRACT from SOURCE as many more words as are in TERM



IF SOURCE word ≠ TERM

ASSUME - if term spans more than 1 page, page number associated w/ term will be the page it ended.

EOF means end of SOURCE

Figure 3, continued

```
SEARCH-SOURCE-FOR-TERM
10 Read in next word from source
   WHILE WORD FROM SOURCE ≠ PH.1 ST WORD IN TERM (AND NOT EOF) DO
      READ IN NEXT WORD FROM SOURCE(J)
      I=J
   END-WHILE
   IF NOT EOF THEN
                GO TO PROCCOMPARE
                GO TO 10
   ENDIF
   RETURN
```

Figure 3, continued

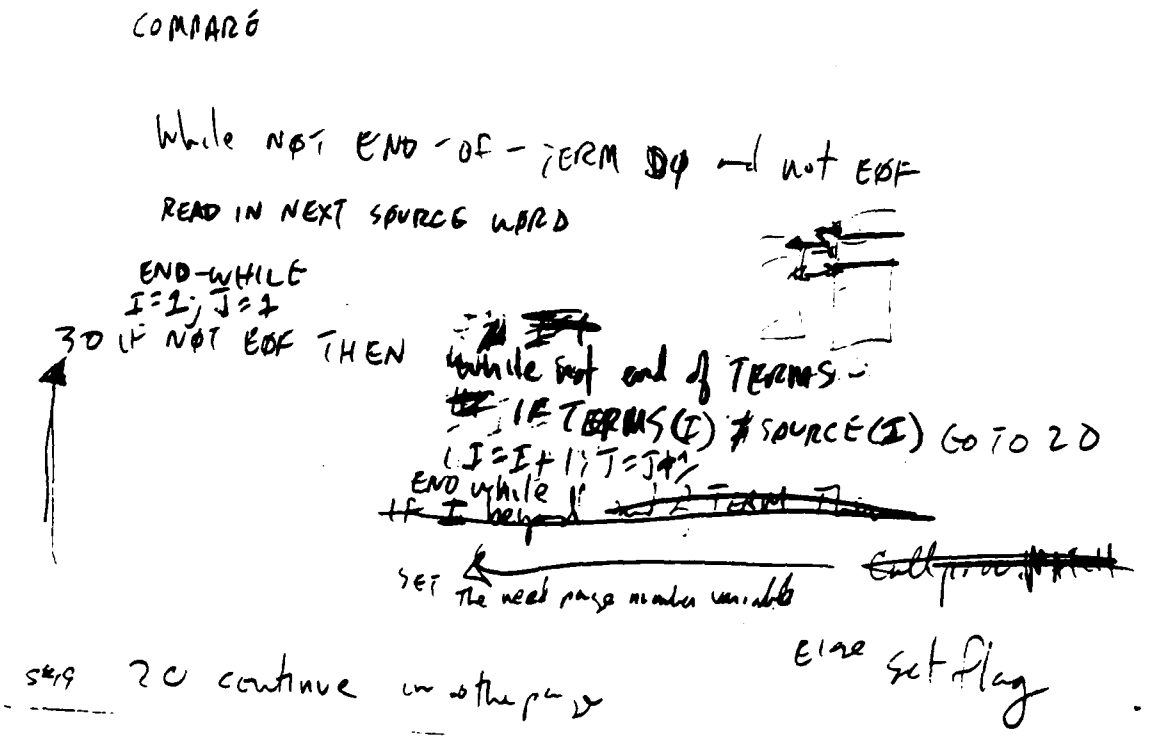


Figure 3, continued

10 Continue

READ NEXT WORD FROM SOURCE
IF got page number and NEED-number then CALL INSERT-page-number
while (word from SOURCE ≠ 1ST WORD IN TERM AND (NOT EOF) DO
READ IN NEXT WORD FROM SOURCE
IF got page number then if need-number then call insert-page-number
END-WHILE READ IN NEXT WORD ~~SPACE~~ from SOURCE

IF NOT EOF THEN ~~EXIT~~ CALL COMPARE
IF EOF EXIT
GO TO 10

ENDIF

RETURN
END

---

20 continue (from COMPARE)
C SOURCE (I) ≠ TERM(I)
J=2; I=1;
IF SOURCE(J) ≠ TERM(I) Then J=J+1 (up to the end of term
GO TO IF
ELSE I=I+1
GO TO 30

Figure 3, continued

stuff" must be done, while something else will be done if a mismatch occurs. This loop is to be repeated until all terms have been dealt with. In reviewing this design (107-115), S19 decides that the SEARCH SOURCE module needs more attention.

S19 expands the SEARCH SOURCE module (117-165) to read the text a line at a time, extracting a buffer of four [sic] words to be compared to a term. The first words of the term and of the text buffer are to be compared. On a mismatch, the compare continues with the next text word, while on a match the rest of the term must be compared to the next words in the text. Once a complete match has occurred, the page number must be isolated and added to the list of page numbers associated with that term. In the course of considering how the later words of the term are compared, S19 notices that the occurrence of a term in the text could cross a line boundary or a page boundary (136-154). He assumes and even records on paper that terms must occur on a single page.

Next, S19 rereads the specifications (166-175). He notes that words do not cross page boundaries, and that each page of text is a single record. He then expands the GET NEXT TERM module by defining a data structure to hold the term and its associated page numbers (180-192). The data structure he chooses is an array, with the first five elements each to hold one word of the term and the remainder to hold page references. So as to be able to specify a maximum size for this array, S19 arbitrarily limits the size of the book to 100 pages total.

S19 goes on to reconsider the SEARCH SOURCE module. He begins by reading the text a record (page) at a time, but abandons this because he doesn't know how many words to expect on a line (211-217). He decides instead to read the text word by word, stripping off all punctuation except for periods.

The SEARCH SOURCE module will cycle through the words of the text, exiting if a match to the first word of the term is found or if the end of the text file is reached (218-130). If the first word of the term is matched, the rest of the term must be tested (231-253). S19 first attempts to reuse the SEARCH SOURCE algorithm on the succeeding words of the term, but he quickly realizes that this will succeed if any subsequent text word matches the term word, rather than just the immediate successor. Instead, he extracts as many words from the text as there are words remaining in the term in order to compare the two sets word by word.

While working out the details of the subsequent comparisons, S19 deals with two special situations. First, he determines that a mismatch should be counted if a period occurs in a potential term (260-276). Second, if a mismatch does occur during these comparisons, the search for additional first word matches should not continue from the last word compared, but from the earliest word of the text that has not been compared to the first word of the term (285-308). This is to be resolved by keeping an index that points to the word from which the search should continue.

S19 decides that the comparison of subsequent term words to the text is complex enough that it should be a separate module (309-357). This COMPARE procedure will generate a buffer of as many text words as there are words in the term and will compare each remaining term word to the equivalent word in the buffer. If the entire term matches successfully, another procedure will be called to insert the page number into the term array. If a mismatch occurs or the end of the text file is reached, a flag is set and control returns to the SEARCH SOURCE module.

While considering what should happen after a match has been found, S19 realizes that he has some problems with the control structure of the top level module (358-372). The initial test for a term match terminates on either a match or the end of the text file. However, these two conditions are to be treated differently; on an end of file, the next term must be accessed, while after a match, the test loop should continue. S19 corrects this by adding GO TO statements that transfer control to the appropriate procedures.

Next S19 considers the MATCH module, which searches forward in the text looking for a page number (373-381). Once the page number is found, it will be inserted in the first empty location of the term array, which is to be flagged in some way.

S19 now describes a procedure to output the term and associated page numbers (382-402). He briefly considers the need to sort the page references in ascending order, but a review of the specifications tells him that they are already correctly ordered.

S19 next reviews the design as it exists thus far (403-602). In doing so, several problems are noticed and modifications made to the design. First, S19 notices again that terms can straddle a record boundary (434-445). He apparently is completely unaware that he resolved

this question once before; this time he decides that terms will be permitted to cross page boundaries, but that the ending page number will be associated with the term.

Reconsideration of the MATCH procedure, which was only sketchily described earlier, leads S19 to decide that those actions (which include reading the text) can better be done within the SEARCH SOURCE module (467-483;503-524). Instead of calling the MATCH routine, the COMPARE procedure simply sets a flag indicating that a page number is needed. If that flag is set, SEARCH SOURCE is to search for an asterisk instead of a match to the first term word.

Another problem is discovered in the COMPARE module (529-540). S19 notes that if the text buffer does not match the term, the words in the text buffer must still be compared to the first word of the term. This cannot be done in the main loop, as it selects words to compare by reading them from the text file. S19 continues reviewing the SEARCH SOURCE module, then returns to work out this comparison in detail (574-607). He decides not only to look for another text match to the first word of the term, but also to search for a complete match if a first word match is found among the items in the text buffer. He gets embroiled in how to advance his indices and decides to abandon the routine because he is pressed for time, but that with enough effort it could be made to work.

The rest of the protocol (608-711) is a review of the design for the experimenter's benefit.

Results from the Encodings

Encoding the protocols of S17 and S19 into rule form brought out several common aspects of their design behavior. We will consider below how each of the sources of information found in the rules contributes to the novices' solution of a design problem. Policies, because they are the general strategies that a designer uses to solve many problems, provide the most direct evidence of the problem solving processes used to solve such problems. Goals, on the other hand, correspond more closely to the results of applying particular problem solving strategies, and notes represent the data that those processes employ and that they produce.

Policies

Unlike the expert designers, novices do not make extensive use of policies. For S17, 6% of the rules have a policy as one of their activating conditions, and for S19, 14% do. Compare this to S3, one of our expert subjects, for whom about 19% of the rule instantiations were guided by policies.

Furthermore, the use of policies by the novices is quite idiosyncratic. The only policy identified for S17 was to *be efficient*. A desire for efficiency motivates a number of his decisions. However, in at least one situation where concern for efficiency would substantially improve the design (the selection of a data structure for the terms), there is no discussion of such issues, and the decision finally made is very inefficient. Apparently, S17 does not fully understand the implications of this term. He understands that efficiency is desirable, but is not completely sure how to achieve it. We did not attribute a policy to be efficient to S19, despite the fact that he frequently brings up the topic. His focus on efficiency appears to be limited to noticing, after decisions are made, that aspects of his design are suboptimal. We found no case where a design decision of S19 was constrained by concern for efficiency.

S19 is equally idiosyncratic in his use of policies. We identified two policies that he used. At one point we attribute to him a policy to *do initial housekeeping*, as he mentions that it is the standard thing to do at the beginning of a module, although he does not mention it in connection with any other module. The other policy, to *focus on iterations*, S19 uses relatively often. Dealing carefully with the control structure of a module (i.e., types of control constructs, termination conditions) forces a designer to pay attention to the boundary conditions of various actions. This strategy is often used by experts as a way of assuring that a portion of the design performs as desired. It is possible that S19 has assimilated this particular aspect of expert design behavior. However, an alternative explanation is that he has learned that iterations are a frequent source of errors for him. This concern for control structures may well reflect his attempts to minimize these errors.

That novices do not have a large store of policies is not particularly surprising. Policies represent a designer's collection of "design lore". For the most part, they are not strategies that are explicitly taught, but are acquired gradually by experience in generating designs.

Our subjects simply have not had enough experience to have assimilated a set of global strategies that can assist them in producing designs. In our study of experts (Atwood and Jeffries, 1980), we found that policies were used to guide the generation and evaluation of design alternatives, to prescribe functions that design components must satisfy, and to aid in determining what elements of the design to consider next. Because the novices do not have policies to guide them in these endeavors, they generate and evaluate fewer alternatives, they are less methodical in their expansion of design components, and they often overlook important functions that must be accomplished.

Goals

Goals are used in our encodings to indicate the actions that a designer intends to accomplish to produce the required design. By examining such statistics as when goals were created and satisfied and the hierarchical relationships among active goals, we can extract information about the processes novices use to keep track of what they are doing and of what remains to be done.

Figure 4 gives a portion of the *goal stack* for S19.This figure shows which goals were active at the time each rule was invoked and which goals were satisfied after the invocation of that rule. Figure 5 gives a *goal hierarchy* for the same goals; it shows the subgoal/supergoal relationships among these goals and indicates the order in which they were activated.

What is most apparent from examination of the goal stack is that the number of active goals varies quite systematically across time. The active goals gradually increase, until a relatively large number of them are satisfied simultaneously or in close succession. Then, the goal stack begins to increase again. At these points of relative minima, the protocols consistently indicate a shift in focus from one aspect of the design to another. This is true of both expert and novice designers.

At this level, the dynamics of goal stack management are similar for both novice and expert designers. The fact that the number of active goals is constrained is not surprising. The goal information, that represents a designer's understanding of the current state of the design, must be managed so that it remains within the capacity of the available working

event	active goals	goals satisfied
25.	28	
29.	28, 16	
31.	28, 16, 1	
32.		28, 16, 1
34.	18	
35.	18, 26	
39.	18	26
40.	18, 19	
41.	18, 19, 24	
43.	18, 19, 24, 14	
45.	18, 19, 24, 14, 13	
47.	18, 19, 24, 14	13
48.	18	19, 24, 14
49.	18, 23	
50.	18, 23, 27	
52.	18, 23	27
53.	18, 23, 6	
55.	18, 23, 6, 9	
58.	18, 23, 6, 9, 2	
60.	18, 23, 6, 9, 2, 4	
65.	18, 23, 6, 9, 2, 4, 3	

Figure 4: Example of a Goal Stack

memory.

Although the dynamics of goal stack management are overtly similar, the processes underlying this management do differ as a function of design expertise. This can be illustrated by comparing the goal stacks with the hierarchical arrangement of goals; that is, by considering why goals were created and how they are satisfied.

In order to accomplish the goals they have generated, our subjects often find it necessary to achieve a sequence of subgoals. Figure 5 is an example of the types of relationships that occur between successive goals. Detailed examination of this figure together with the goal stack of Figure 4 enables us to infer that the active goals of novice designers are organized into a very simple structure. Goals are enumerated in a purely depth-first manner. A goal is invoked, followed by one or more generations of subgoals. Satisfaction of the lowest level subgoal causes its ancestors to be satisfied, followed by the invocation of a new subgoal at one of the upper levels. Except for the top few levels, goals almost invariably have only one

```
 7-28.  review module just written
   8-16.  determine what to do if whole term found
     9- 1.  associate page numbers with term
10-18.  expand abstract design
   11-26.  reread problem specifications
   12-19.  expand GET NEXT TERM module
      13-24.  expand term data structure
         14-14.  determine size of term array
            15-13.  determine maximum number of page numbers
                    associated with term
   16-23.  expand SEARCH SOURCE module
      17-27.  read source
      18- 6.  consider case of word of source not equal to
              first word of term
         19- 9.  consider termination of compare-read loop
            20- 2.  consider case of first word match
               21- 4.  consider case of subsequent word
                       compares
                  22- 3.  consider case of mismatch
```

Figure 5: Example of Goal Structure

[Note: The first number indicates the relative order of goal activation; the second number identifies the goal (i.e., is the same as is used in Figure 4).]

offspring; that is, a goal seldom produces two or more subgoals at the same level. In those cases at the higher levels where a goal activates sibling offspring (e.g., the goal to *expand abstract design* leads to the generation of a subgoal for each module in the abstract design), each goal at the lower level is resolved before the next sibling is considered.

This simple goal management strategy is in sharp contrast to the complex goal hierarchies generated by experts. Expert goals are likely to have multiple sibling subgoals at all levels. Furthermore, the set of sibling goals is frequently enumerated when the subject first begins to work on the higher level goal. For example, the expert may articulate a goal to *consider termination conditions for module A*, then enumerate the tests to be considered, then explore each one in a depth-first manner. In contrast, the novice would only explore in detail multiple test conditions if the parent goal were one of the highest level goals and, furthermore, would resolve each condition before going on to consider the next one.

The experts exhibit other complexities in their goal management. They are able to interrupt a goal with a complex digression, explore a peripherally related topic in depth, and return unerringly to the original subject. They also occasionally *defer* goals; i.e., decide that

an issue cannot be resolved at that time, and keep it in abeyance for several tens of minutes, until new relevant information becomes available. We saw no evidence of the use of such complex strategies in novices.

The differences in the goal management strategies between experts and novices are clear evidence of the novices' reduced ability to deal with the complex processing requirements of the task. However, the cause of the experts' improved performance is not clear from these data. One possible interpretation is that because of improvements in their ability to handle other aspects of the design process, experts can simply devote more of their memory capacity to storing pending goals. Another is that experts develop chunking strategies that enable them to store more of the necessary goal-related information in long term or even external memory. For example, if decisions about the order in which to attack the highest level elements of a design problem are stored as a policy, this would free working memory to encode more of the intermediate goals; e.g., a deferred subgoal, or an interrupted goal.

Notes

Notes represent the information the designer uses to construct the design. Most of the information encoded as notes was derived by the subject during the course of problem solving and is related to particular goals. That is, the creation of a goal leads to the generation of information that is relevant to the satisfaction of that goal. Such information is explicitly generated during the course of design construction. Examples are that *page numbers will be stored in a vector of the term matrix* and *a word of text is to be compared to the first word of the term.*

The other notes are retrieved or inferred from a subject's memory. They come from several different sources: e.g., the *executive*, which monitors the current state of working memory, from *simulating* the effects of the actions of a segment of the design, or from subjects' knowledge of *computer science* or *textbooks and indexing.* Examples of notes from such categories are in Figure 6.

For novices, notes are used to indicate solutions to goals rather than constraints that solutions must satisfy, as they do for expert designers. Consider, for example, S19's goal to

- EXECUTIVE
  - case of word being page number has been considered
  - abstract design is complete
- SIMULATING EFFECTS OF DESIGN
  - current data structure does not allow for page count
  - matches will occur before the page number is found
- COMPUTER SCIENCE
  - * is used in FORTRAN to denote multiplication
  - dimensions of array must be known
- TEXTBOOKS AND INDEXING
  - index terms may appear multiple times
  - source (text) file may be large
- PROBLEM SPECIFICATIONS
  - pages are not in consecutive order
  - source is an ASCII file
- INFERENCES FROM PROBLEM SPECIFICATIONS
  - number of page numbers is not known
  - specifications do not mention size of machine
- DESIGN
  - worksheet contains "note page in terms vector"
  - worksheet contains "if not found on page, continue"

Figure 6: Examples of Notes

expand GET NEXT TERM *module*. Similar goals were also considered by S17 and the expert, S3. For S19, this goal leads to such notes as *read term into data structure* and *make data structure an array*. Likewise, S17 notes to *insert term in data structure*, having decided earlier that *terms will be stored in a vector*. In contrast, S3 generates such facts as *table should have one entry per term, there will be a pointer to each term*, etc. In general, experts use notes to store or record intermediate results, but novices generate far fewer intermediate results, and thus use notes primarily to record final decisions.

The solution-posting, rather than constraint-posting, function of novice's notes is also apparent in comparing the notes used by novices to the experts' goals. Many times an issue that is resolved with a single note in a novice protocol will be expanded into several levels of subgoals by the expert. For example, the novices' resolution of the *expand* GET NEXT TERM *module* goal described above is to simply note that terms are to be stored in a data structure. S3, on the other hand, generates and expands a goal to *consider routine to add items to list*. Similarly, novices select data structures very quickly, while S3, like other experts, spends a great deal of time on the goal to *determine form of term table data structure*.

The ways in which novices use notes are another indication of the impoverished environment under which they must operate. Rather than using notes to accumulate data which they can use to make decisions, novices typically make a decision as soon as an issue arises. This may be that they do not have the knowledge that would enable them to choose among alternatives, even if they had the data relevant to those choices, or it may be that they cannot reasonably retain additional information in working memory, and thus there is no point in attempting to store it.

Novice Design Methods

We turn now from a consideration of the individual design decisions made and the kinds of information contained in those decisions, to an examination of the content of particular problem solving episodes and the information this gives us about the design strategies of the novices. We find that novices differ from experts in two general areas: they only partially understand many concepts in computer science, and they are less able to

execute the basic operations of software design.

Partial Understanding of Concepts

The novices have a very limited understanding of some of the basic concepts of computer science. Their knowledge strongly reflects their experience with particular computers and computer languages. One serious confusion is in the difference between a *computer word* and an *English word.* On the computer that these subjects are most familiar with (Control Data 6400), a computer word can contain an English word of up to ten characters. Thus, for the simple classroom type exercises that students typically do, equating the two concepts does not have serious practical consequences. The misunderstanding leads S19 astray, however, when he attempts to read the text file. He is unable to read the text a page at a time, since he does not know how many (computer) words to allocate, because he does not know how many (English) words there are on a page.

Another concept that is partially misunderstood is that of a data record. These subjects, undoubtedly due to their experience, tend to equate *record* with *card image.* Thus, S17 assumes a maximum of 80 words possible on one line, presumably because a card has 80 columns. This assumption about the "natural" size of a record even overrides the statement in the specifications that a page of text is a single record. Both subjects want to read the text a "word or line at a time", even though S19, at least, specifically notices the record size in the specifications.

The compare operation in the novices' designs also suffers from their limited understanding of such procedures. S19 treats the comparison of a text word to a term word as equivalent to comparing two numbers. However, it is not clear how much of this to attribute to his misunderstanding of what a word is and how much to misunderstanding the compare operation itself. Neither subject is concerned with any of the complexities of the compare operation - cases of the letters, punctuation, how to minimize the number of comparisons needed, etc. (S19 does consider how punctuation affects the match, but his solution is wrong). S17 resolves all potential problems by saying "do it in SNOBOL", as SNOBOL is a language for which such issues are at least partially transparent to the

programmer.

The misunderstandings that these subjects had in the examples above can not be attributed to a lack of exposure to the correct form of the concept. All of these constructs (character comparison, character data structures, record input and output) had been covered in the course from which these students were recruited; furthermore, they had undoubtedly been discussed in other computer science courses the subjects had taken. It is very likely that had we asked the students directly about some of their misunderstandings (e.g., will a capital A match a small a?), they would have answered correctly. However, in an actual problem solving situation they were unable to apply knowledge they almost surely had. Apparently the cues generated by the problem specifications and the task environment were not sufficient to retrieve important, relevant information about a concept.

Another example of the novices' failure to recall relevant information can be seen in their selection of a data structure for the terms. Both subjects had been exposed to the concept of a linked list, probably several times. However, although this is an eminently suitable situation in which to use such a structure, both subjects chose to store the terms and associated page numbers in an array. They are aware that this data structure consumes a large amount of storage -- S19 notes that his choice is "inefficient", and S17 limits its size by restricting the size of the book to be indexed. However, neither of them reconsiders his decision about the form of the data structure. Their knowledge about linked lists is simply not seen as relevant to the current situation.

Execution of Software Design Operations

Earlier we defined software design as consisting of three primary operations: 1) modular decomposition, 2) specifying the interactions among modules, including the control structures, and 3) specifying the data structures. One might also include as an important design operation the ability to critique the design and to understand the criteria by which it should be evaluated. Novices have difficulty carrying out each of these activities.

On the initial pass over the design, novices decompose the problem as well as the experts do. We attribute this to the fact that this particular problem has a straightforward

solution model, equally derivable by solvers at all experience levels. By solution model, we mean the initial breakdown of the problem into a small number of loosely defined subproblems that comprise the designer's internal representation of how to attack the problem. All of our designers, both experts and novices, had approximately equivalent solution models and were able to refine these models into an adequate first level decomposition. The novices at times omitted or incorrectly dealt with particular details at this level, but the end result did not differ qualitatively from that of the experts. It is in their attempts to carry the decomposition process to lower levels that the novices break down.

S17 is perfectly content to consider the design complete at this first level; it is only at the experimenter's urging that he continues. He at no point breaks down any of his modules into additional modules. His further iterations can be seen as adding *steps* to the program -- refining the data structure, incorporating tests for special conditions such as end of file, deciding how input is to be done. He begins each pass by reviewing the current state of the design. This causes him to notice new facts: e.g., that the page number is not known when a match occurs, or that a module does not have a terminating condition. The design is augmented to deal with these additional pieces of information.

S19 is somewhat better at decomposing. He takes the design through several passes without prompting from the experimenter. He not only adds information to modules, but he specifies what is to be done in more detail. For example, on the first pass, he intends to search the text for the term, and "do some kind of stuff" on a match. This is expanded to first search for a match between the text and the first word of the term, then to compare the rest of the term to the text word by word. On the next iteration, he implements counters that keep track of this compare process and also adds statements that enable him to "get back on track" should the compare fail in the middle of the term. These expansions are not generally done by decomposing the module into submodules (although he does do that in one case), but each pass specifies the same actions at a greater level of detail, as well as adds new actions.

S19 has difficulty modularizing these lower level expansions, because the actions frequently interact. At one point his SEARCH SOURCE FOR TERM module searches for a

match to the first word of the term, then calls a procedure COMPARE to compare the rest of the term to the text. COMPARE calls a procedure MATCH to store the page number when a successful comparison occurs. When S19 goes to expand the details of the MATCH module, he discovers that he cannot store the page number until the entire page has been read. With this complication, SEARCH SOURCE and MATCH interact; S19 is able to avoid dealing with the consequences of this interaction by incorporating the actions of the MATCH procedure into the SEARCH SOURCE module. This example and other evidence suggest that S19 will decompose modules into submodules when the tasks are highly separable, but he does not have available the methods experts use to minimize the interactions among segments and to communicate between modules when the interactions cannot be eliminated.

Turning to the second aspect of design, dealing with interactions, we see little evidence concerning the novices' ability to deal with interactions between modules, because they do so little decomposing beyond the first level. For this particular problem, the top level decomposition has a very straightforward control structure that leads to few interactions. In the one situation in which S19 encountered an interaction between modules, he combined the two tasks into a single module. Similarly, when S17 discovers late in his effort that terms can be phrases, instead of only single words, he seems completely oblivious to the major modifications that must be made to his design to accommodate this change. He choses merely to make this information available to the implementing programmer, because of its potential effect on the data structures.

The novices do have some strategies for dealing with interactions within modules. They have clearly been taught structured programming concepts (Dahl, Dijkstra, and Hoare, 1972) and try to apply them to the control structures within each module. S17 describes his control structures so sketchily that it is hard to extract any generalizations, but S19 shows evidence of having consistent problems with the flow of control in his design.

S19 pays careful attention to his control structures, perhaps because he is aware that they are a source of difficulty for him. For each action he must perform, he considers the conditions that must be true for that action to occur, and embeds the action in a control construct (IF/THEN, WHILE) with the appropriate tests. In contrast to the typical novice

programmer's failure to test necessary conditions, S19 overdetermines his conditions. Frequently he tests for the truth of condition A in a branch that will only be executed when A is true. A more subtle example occurs in his top level loop. The main part of this module can be sketched as follows:

```
WHILE not end of text file
      AND not match to first word of term DO
          look for first word match;
 IF not end of text file
     THEN compare additional words;
```

S19 notices on a subsequent pass that after searching for additional word matches after the initial match, the design terminates, instead of searching for more first word matches. He realizes that the problem is due to the two tests in the WHILE loop having entirely different consequences -- one should initiate search with a different term, while the second should cause a compare procedure to be executed and search with the current term to continue. His correction is to add a GOTO statement, immediately after the compare, that passes control to the beginning of the WHILE loop. This results in intertwining two activities that should be considered separately -- the search for a match and the iteration over the text file. The two actions can be quite clearly separated with, for example, the following control structure:

```
WHILE not end of text file DO
    IF first word match
        THEN compare additional words;
```

S19's problems may lie in the fact that he designs his control structures concurrently with his decompositions (more correctly, iterations). Our experts, at least on the more difficult phases of the design, isolated the actions to be performed first and then attempted to embed them in the appropriate control structure. Interactions such as the one described above cannot occur when the different functions are clearly separated into distinct modules. S19's inability to isolate separate functions interferes with his attempts to decompose and with his attempts to control the flow of activity.

The novices also had difficulty with the third aspect of design, constructing appropriate data structures. The confusion over the meaning of "word", discussed above, led S19 to store the term as an array of five words, rather than as a character string. Neither subject attempted to impose any structure on the list of terms so as to make comparison with the text more efficient. And both novices stored the page numbers associated with each term in a

large array, rather than some sort of list structure. All of these errors point to the novices' lack of understanding of certain concepts in computer science. In addition, the method these designers used to construct the data structure led them into difficulty.

S17's method of building the term data structure can best be described as "on the fly". He first notices that he will need a place to store the terms and allocates a vector for this purpose. When he first considers storing the page numbers, he discovers he needs a place to put them, so the vector becomes a matrix, with the first column to hold the terms and the remaining ones to hold page numbers. Later he realizes that he needs an index to tell him where in each row to insert the next relevant page number; he then adds a column to his matrix to serve this function. When he discovers that the page number will not be known when the match occurs, he reserves another column of his matrix to serve as a count (flag) of whether the term was found on the current page. On the next pass over the design he decides to consolidate this information and defines the matrix as containing the terms, the "count", and the page numbers, but not the index. When he goes to store the page number, he notices the oversight and patches it in.

The construction of this data structure is completely driven by particular storage requirements that are discovered while iterating over the actions of the design. As each need is uncovered, it is added piecemeal to the current data structure, resulting in a "patchwork quilt" solution. No attention is paid to possible interactions among items. As it happens, no interactions occur, so we cannot be sure that S17 would not have noticed and corrected any such problems. However, since he was unable to even remember all the components at once, it does not seem unreasonable to infer that he would have had difficulty noticing problems involving relationships among elements.

An understanding of the properties the finished design should display and of some heuristics for deciding when an effort is "close enough" to those ideal properties is an important component of a designer's skill. The novices have developed some ability to critique their designs, but the strategies they use to ensure that the designs meet these criteria are very different from those used by experienced designers. As an example, S17's rules for deciding when a design has sufficient detail are quite different from those of the

experimenter and of our expert subjects. As another case, S17 and S19 both expressed a desire for efficiency in their designs, but they had a great deal of difficulty deciding what is needed to make a design efficient and carrying out those decisions. While both subjects ran into trouble in this area, they had very different ways of dealing with the efficiency issue.

S19's concern with efficiency is limited to critiques of his decisions. He frequently makes comments like: "it's inefficient and expensive, but it's easy", or "there might be a better way to do that", but little of this noticing leads to changes in the design. He seems to be doing something more than just paying lip service to the goal of efficiency, but he is unable to come up with more efficient alternatives to his original decisions.

S17 has a little more sophisticated ability to improve the efficiency of his design. He both notices some inefficient constructs and attempts to improve them. His repertory of methods for increasing efficiency is quite limited, and he is especially hampered by his incomplete understanding of some of the concepts he is working with. For example, as was mentioned above, when he notes that his data structure for the terms consumes a large amount of storage, he limits the size of the array, rather than converting it to a linked list or some other more compact structure.

Another example comes from his consideration of whether or not to read and store a full page of text. If the current page is kept in memory, S17 will need to allocate storage for it, while if he reads the text a word at a time, he will need to allocate a flag for each term to keep track of which term lists to add the page number to. He chooses the latter course on the basis of "efficiency" (presumably storage efficiency), even though a rough comparison of the number of terms to be expected in an index versus the amount of storage needed for a single page of text should lead one to prefer storing the page of text. S17, however, does not attempt this calculation.

Experienced software designers consider efficiency to be a useful criterion by which they judge their designs. S17 and S19 have learned that this is an important dimension along which to evaluate their work, but they have not yet acquired the tools by which to do this successfully. We see no evidence from S19 of any method for implementing the results of his critiques. S17, in contrast, has developed the rudiments of a method for generating

alternative solutions and evaluating them on an efficiency basis. He does not do so consistently (i.e., he misses the components of the design where efficiency would have the greatest impact), and his attempts are hindered by his inability to bring the relevant information to bear. Again, the knowledge that he must surely have (e.g., the ability to calculate the relative space requirements of the two alternatives) is not available in the context in which he needs it.

In our earlier report considering expert design behavior, we noted three characteristic skills that experts employ. They retrieve previously developed solutions and modify them to fit the current context (problem solving by debugging almost-right plans), they do not backtrack, in the sense of making design modifications that would propagate across several components, and they are able to retrieve previously generated information whenever it is relevant.

In the novice protocols, the opposite is true for each of these attributes. Although novices do retrieve previously developed or learned solutions, they make no attempt to modify them to fit the current design effort. Solutions are used exactly as retrieved. Further, these solutions are at a fairly low level of detail. In considering how to compare text and terms, for example, S3 attempts to modify the process used to verify two files. Novices, on the other hand, apparently attempt to apply the same procedure that would be be used to compare numeric values, as opposed to character strings, without modification. Novices retrieve information on how to construct arrays or vectors or how to apply primitive design operations; higher-order solutions are not retrieved.

Experts are very reluctant to rewrite a module, if problems with that module are discovered, since the changes may affect other aspects of the design. S19, on the other hand, frequently rewrites the COMPARE module, without considering how these changes affect the remainder of the design.

The novices' ability to remember what they have done already in the problem solving session is so limited that they sometimes have to solve the same subproblem several times. S19 notes, very early in his design effort, that a term may straddle a page boundary. This is an important aspect of this design, and one frequently overlooked by expert designers. He

records, externally, his assumption that this will not happen. Later, he notices this problem again, treating it as a new discovery; no mention is made of the fact that this was considered earlier. Similarly, S17 notices that his design does not handle multiple word terms, as is specified. He comments on the corrections that would be required, but does not incorporate them. Late in the design effort, he discovers this fact again. Experts, on the other hand, are better able to recall relevant design information that was generated previously.

In summary, there are striking differences between the ways expert and novice software designers attack the same subproblems. For the novices, the indexer problem is quite hard, and they find it difficult to marshall the resources needed to solve it. In contrast, the experts have access to a host of specialized procedures that make solving this problem much easier. In the next section, we will examine the kinds of general processes the novices use in the absence of these specialized skills.

A More Global View

What can we say about the way in which the novices approach the design task? In an earlier paper (Jeffries, Turner, Polson, and Atwood, 1981), we described the design schema, a specialized collection of knowledge and procedures that experts use to manage design activities. We see no evidence that the novices have anything more than the rudiments of such a schema. Their approach appears to be much more closely akin to the general tools they might use in many problem solving situations. In fact, much of it looks like means-ends analysis.

A great many of the goals produced by the novices are generated in an attempt to reduce some difference -- e.g., supply some missing information, specify the result of some operation. Design is an instance of an extremely ill-structured problem and as such does not easily lend itself to a description in terms of *states* and *operators*. Nonetheless, many of the novices' problem solving episodes clearly fit a means-ends analysis framework. A desired state is mentioned (e.g., terms are to be input), obstacles to achieving that state are enumerated (e.g., how to read the terms, where to store them, when to quit), and if possible, overcome. This is in contrast to the typical expert strategy, where a goal is enumerated, it is

broken down into subproblems, and information relevant to solving the subproblems is derived, with solutions to later problems building on earlier results.

Novices attack subproblems by working backward. A good illustration of this can be seen in S19's expansion of the GET NEXT TERM module. S19 has decided to store the term in some data structure, which leads to the subgoal *expand the term data structure.* He chooses an array as his data structure; this produces a subgoal to *determine the size of the term array,* which leads to a goal to find *the maximum number of page numbers that will be associated with a term.* Each goal in the above example was created in order to provide information that is necessary to achieve its parent goal. The expert strategy is more of a working forward approach, where new solutions are derived from prior solutions.

Several examples discussed above can be taken as instances of the working backward approach. The piecemeal design of the data structure is a clear instance. Rather than gathering information as to the functions of and constraints on the data structure and translating that into a particular storage arrangement, S17 constructs this structure completely in response to demands imposed elsewhere in the design and does so at the time that these demands are in focus for other purposes. This approach is also illustrated by S17's second and later iterations over the design. Differences between the design in its current state and the problem specifications or S17's knowledge of indexes (which are at least implicitly part of the problem specifications) lead to goals that resolve these differences by adding steps to the design.

This working backward strategy has been found repeatedly in studies of novice physics problem solving (Chi, Feltovich, and Glaser, 1980). Physics novices retrieve equations because of an explicitly stated unknown. They do not, as experts do, generate other, often more abstract information that could be useful in solving the problem. Our software design novices behave similarly; subgoals are created to reduce differences.

Perhaps related to the working backwards strategy is the novices' tendency to accept the first solution that occurs to them for any subproblem. This was apparent in the rule encodings, where issues that generated multiple subgoals for the expert were often solved with a single note by the novices. The choice of a data structure also fits this description.

While the experts treated the determination of a data structure as a difficult task, both novices immediately selected an array without any examination of what effect their choice might have on the rest of the design. S19, at least, was not satisfied with this decision, but his dissatisfaction was not enough to cause him to modify his choice.

It appears that the working backwards approach may impose this immediate solution strategy. Experts use the constraining information that they generate as a matter of course during the solution attempt to decide among alternative actions. Novices are interested in reducing differences; because they do not collect this additional information, they have no data from which to prefer one way of reducing a difference over another.

This is not to say that all of the novices' solution techniques are completely domain-independent. They clearly do bring to bear relevant computer science information to the solution of this problem. Furthermore, there is evidence that they are developing the rudiments of the same design schema we attribute to experts. Both novices were able to effect the first level decomposition of this admittedly straightforward problem. S19 was able, in some cases, to take this decomposition process to an additional level. Both subjects are learning some of the evaluative criteria that characterize good designs, and S17 has learned a few methods for modifying his designs to meet those criteria. While these subjects are far from accomplished software designers, it is apparent that they have assimilated some of the "secrets" of this trade. Presumably, with additional experience, less of their behavior will be controlled by general, default problem solving processes, and more by the control schema that mediates expert performance in this field.

References

Atwood, M.E., and Jeffries, R. *Studies in Plan Construction I: Analysis of an Extended Protocol.* Technical Report SAI-80-028-DEN, Science Applications, Inc., 1980.

Chi, M.T.H., Feltovich, P.J., & Glaser, R. *Representation of physics knowledge by experts and novices.* Technical Report 2, University of Pittsburgh, Learning Research and Development Center, 1980.

Dahl, O-J., Dijkstra, E.W., and Hoare, C.A.R. *Structured Programming.* New York: Academic Press, 1972.

Jeffries, R., Turner, A.A., Polson, P.G., and Atwood, M.E. Processes Involved in Designing Software. In J. A. Anderson (Ed.), *Cognitive Skills and Their Acquisition,* New York: Lawrence Erlbaum and Associates, 1981.

1 Meryl S. Baker
NPRDC
Code P309
San Diego, CA 92152

1 Dr. Jack R. Borsting
Provost & Academic Dean
U.S. Naval Postgraduate School
Monterey, CA 93940

1 Dr. Robert Breaux
Code N-711
NAVTRAEQUIPCEN
Orlando, FL 32813

1 Dr. Richard Elster
Department of Administrative Sciences
Naval Postgraduate School
Monterey, CA 93940

1 DR. PAT FEDERICO
NAVY PERSONNEL R&D CENTER
SAN DIEGO, CA 92152

1 Dr. John Ford
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Henry M. Halff
Department of Psychology,C-009
University of California at San Diego
La Jolla, CA 92093

1 LT Steven D. Harris, MSC, USN
Code 6021
Naval Air Development Center
Warminster, Pennsylvania 18974

1 Dr. Jim Hollan
Code 304
Navy Personnel R & D Center
San Diego, CA 92152

1 CDR Charles W. Hutchins
Naval Air Systems Command Hq
AIR-340F
Navy Department
Washington, DC 20361

1 CDR Robert S. Kennedy
Head, Human Performance Sciences
Naval Aerospace Medical Research Lab
Box 29407
New Orleans, LA 70189

1 Dr. Norman J. Kerr
Chief of Naval Technical Training
Naval Air Station Memphis (75)
Millington, TN 38054

1 Dr. William L. Maloy
Principal Civilian Advisor for
Education and Training
Naval Training Command, Code 00A
Pensacola, FL 32508

1 CAPT Richard L. Martin, USN
Prospective Commanding Officer
USS Carl Vinson (CVN-70)
Newport News Shipbuilding and Drydock Co
Newport News, VA 23607

1 Dr. James McBride
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr William Montague
Navy Personnel R&D Center
San Diego, CA 92152

1 Library
Naval Health Research Center
P. O. Box 85122
San Diego, CA 92138

1 Naval Medical R&D Command
Code 44
National Naval Medical Center
Bethesda, MD 20014

1 Ted M. I. Yellen
Technical Information Office, Code 201
NAVY PERSONNEL R&D CENTER
SAN DIEGO, CA 92152

1 Library, Code P201L
Navy Personnel R&D Center
San Diego, CA 92152

1 Technical Director
Navy Personnel R&D Center
San Diego, CA 92152

6 Commanding Officer
Naval Research Laboratory
Code 2627
Washington, DC 20390

1 Psychologist
ONR Branch Office
Bldg 114, Section D
666 Summer Street
Boston, MA 02210

1 Psychologist
ONR Branch Office
536 S. Clark Street
Chicago, IL 60605

1 Office of Naval Research
Code 437
800 N. Quincy SStreet
Arlington, VA 22217

5 Personnel & Training Research Programs
(Code 458)
Office of Naval Research
Arlington, VA 22217

1 Psychologist
ONR Branch Office
1030 East Green Street
Pasadena, CA 91101

1 Special Asst. for Education and
Training (OP-01E)
Rm. 2705 Arlington Annex
Washington, DC 20370

1 Office of the Chief of Naval Operations
Research Development & Studies Branch
(OP-115)
Washington, DC 20350

1 Dr. Donald F. Parker
Graduate School of Business Administrati
University of Michigan
Ann Arbor, MI 48109

1 LT Frank C. Petho, MSC, USN (Ph.D)
Code L51
Naval Aerospace Medical Research Laborat
Pensacola, FL 32508

1 Dr. Gary Poock
Operations Research Department
Code 55PK
Naval Postgraduate School
Monterey, CA 93940

1 Roger W. Remington, Ph.D
Code L52
NAMRL
Pensacola, FL 32508

1 Dr. Bernard Rimland (03B)
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Worth Scanland
Chief of Naval Education and Training
Code N-5
NAS, Pensacola, FL 32508

1 Dr. Robert G. Smith
Office of Chief of Naval Operations
OP-987H
Washington, DC 20350

1 Dr. Alfred F. Smode
Training Analysis & Evaluation Group
(TAEG)
Dept. of the Navy
Orlando, FL 32813

1 Dr. Richard Sorensen
Navy Personnel R&D Center
San Diego, CA 92152

1 Roger Weissinger-Baylon
Department of Administrative Sciences
Naval Postgraduate School
Monterey, CA 93940

1 Dr. Robert Wisher
Code 309
Navy Personnel R&D Center
San Diego, CA 92152

1 Mr John H. Wolfe
Code P310
U. S. Navy Personnel Research and
Development Center
San Diego, CA 92152

1 Technical Director
U. S. Army Research Institute for the
Behavioral and Social Sciences
5001 Eisenhower Avenue
Alexandria, VA 22333

1 HQ USAREUE & 7th Army
ODCSOPS
USAAREUE Director of GED
APO New York 09403

1 DR. RALPH DUSEK
U.S. ARMY RESEARCH INSTITUTE
5001 EISENHOWER AVENUE
ALEXANDRIA, VA 22333

1 Dr. Dexter Fletcher
U.S. Army Research Institute
5001 Eisenhower Avenue
Alexandria,VA 22333

1 DR. FRANK J. HARRIS
U.S. ARMY RESEARCH INSTITUTE
5001 EISENHOWER AVENUE
ALEXANDRIA, VA 22333

1 Dr. Michael Kaplan
U.S. ARMY RESEARCH INSTITUTE
5001 EISENHOWER AVENUE
ALEXANDRIA, VA 22333

1 Dr. Milton S. Katz
Training Technical Area
U.S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Harold F. O'Neil, Jr.
Attn: PERI-OK
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Robert Sasmor
U. S. Army Research Institute for the
Behavioral and Social Sciences
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Frederick Steinheiser
U. S. Army Reserch Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Joseph Ward
U.S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Earl A. Alluisi
HQ, AFHRL (AFSC)
Brooks AFB, TX 78235

1 Dr. Genevieve Haddad
Program Manager
Life Sciences Directorate
AFOSR
Bolling AFB, DC 20332

1 Dr. Marty Rockway
Technical Director
AFHRL(OT)
Williams AFB, AZ 58224

2 3700 TCHTW/TTGH Stop 32
Sheppard AFB, TX 76311

1 Jack A. Thorp, Maj., USAF
Life Sciences Directorate
AFOSR
Bolling AFB, DC 20332

1 H. William Greenup
Education Advisor (E031)
Education Center, MCDEC
Quantico, VA 22134

1 Special Assistant for Marine
Corps Matters
Code 100M
Office of Naval Research
800 N. Quincy St.
Arlington, VA 22217

1 DR. A.L. SLAFKOSKY
SCIENTIFIC ADVISOR (CODE RD-1)
HQ, U.S. MARINE CORPS
WASHINGTON, DC 20380

12 Defense Technical Information Center
Cameron Station, Bldg 5
Alexandria, VA 22314
Attn: TC

1 Dr. Craig I. Fields
Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, VA 22209

1 Military Assistant for Training and
Personnel Technology
Office of the Under Secretary of Defense
for Research & Engineering
Room 3D129, The Pentagon
Washington, DC 20301

1 Dr. Susan Chipman
Learning and Development
National Institute of Education
1200 19th Street NW
Washington, DC 20208

1 Dr. Joseph I. Lipson
SEDR W-638
National Science Foundation
Washington, DC 20550

1 William J. McLaurin
Rm. 301, Internal Revenue Service
2221 Jefferson Davis Highway
Arlington, VA 22202

1 Dr. Arthur Melmed
National Intitute of Education
1200 19th Street NW
Washington, DC 20208

1 Dr. Andrew R. Molnar
Science Education Dev.
and Research
National Science Foundation
Washington, DC 20550

1 Personnel R&D Center
Office of Personnel Managment
1900 E Street NW
Washington, DC 20415

1 Dr. Frank Withrow
U. S. Office of Education
400 Maryland Ave. SW
Washington, DC 20202

1 Dr. Joseph L. Young, Director
Memory & Cognitive Processes
National Science Foundation
Washington, DC 20550

1 Dr. John R. Anderson
Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213

1 Dr. John Annett
Department of Psychology
University of Warwick
Coventry CV4 7AL
ENGLAND

1 1 psychological research unit
Dept. of Defense (Army Office)
Campbell Park Offices
Canberra ACT 2600, Australia

1 Dr. Patricia Baggett
Department of Psychology
University of Denver
University Park
Denver, CO 80208

1 Dr. Nicholas A. Bond
Dept. of Psychology
Sacramento State College
600 Jay Street
Sacramento, CA 95819

1 Dr. Lyle Bourne
Department of Psychology
University of Colorado
Boulder, CO 80309

1 Dr. John S. Brown
XEROX Palo Alto Research Center
3333 Coyote Road
Palo Alto, CA 94304

1 Dr. Bruce Buchanan
Department of Computer Science
Stanford University
Stanford, CA 94305

1 DR. C. VICTOR BUNDERSON
WICAT INC.
UNIVERSITY PLAZA, SUITE 10
1160 SO. STATE ST.
OREM, UT 84057

1 Dr. Pat Carpenter
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

1 Dr. John B. Carroll
Psychometric Lab
Univ. of No. Carolina
Davie Hall 013A
Chapel Hill, NC 27514

1 Charles Myers Library
Livingstone House
Livingstone Road
Stratford
London E15 2LJ
ENGLAND

1 Dr. William Chase
Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213

1 Dr. Micheline Chi
Learning R & D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15213

1 Dr. Allan M. Collins
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, Ma 02138

1 Dr. Lynn A. Cooper
LRDC
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15213

1 Dr. Meredith P. Crawford
American Psychological Association
1200 17th Street, N.W.
Washington, DC 20036

1 Dr. Hubert Dreyfus
Department of Philosophy
University of California
Berkely, CA 94720

1 LCOL J. C. Eggenberger
DIRECTORATE OF PERSONNEL APPLIED RESEARC
NATIONAL DEFENCE HQ
101 COLONEL BY DRIVE
OTTAWA, CANADA K1A OK2

1 Dr. Ed Feigenbaum
Department of Computer Science
Stanford University
Stanford, CA 94305

1 Dr. Richard L. Ferguson
The American College Testing Program
P.O. Box 168
Iowa City, IA 52240

1 Mr. Wallace Feurzeig
Bolt Beranek & Newman, Inc.
50 Moulton St.
Cambridge, MA 02138

1 Dr. Victor Fields
Dept. of Psychology
Montgomery College
Rockville, MD 20850

1 Dr. John R. Frederiksen
Bolt Beranek & Newman
50 Moulton Street
Cambridge, MA 02138

1 Dr. Alinda Friedman
Department of Psychology
University of Alberta
Edmonton, Alberta
CANADA T6G 2E9

1 Dr. R. Edward Geiselman
Department of Psychology
University of California
Los Angeles, CA 90024

1 DR. ROBERT GLASER
LRDC
UNIVERSITY OF PITTSBURGH
3939 O'HARA STREET
PITTSBURGH, PA 15213

1 Dr. Marvin D. Glock
217 Stone Hall
Cornell University
Ithaca, NY 14853

1 Dr. Daniel Gopher
Industrial & Management Engineering
Technion-Israel Institute of Technology
Haifa
ISRAEL

1 DR. JAMES G. GREENO
LRDC
UNIVERSITY OF PITTSBURGH
3939 O'HARA STREET
PITTSBURGH, PA 15213

1 Dr. Harold Hawkins
Department of Psychology
University of Oregon
Eugene OR 97403

1 Dr. James R. Hoffman
Department of Psychology
University of Delaware
Newark, DE 19711

1 Glenda Greenwald, Ed.
"Human Intelligence Newsletter"
P. O. Box 1163
Birmingham, MI 48012

1 Dr. Earl Hunt
Dept. of Psychology
University of Washington
Seattle, WA 98105

1 Dr. Steven W. Keele
Dept. of Psychology
University of Oregon
Eugene, OR 97403

1 Dr. Walter Kintsch
Department of Psychology
University of Colorado
Boulder, CO 80302

1 Dr. David Kieras
Department of Psychology
University of Arizona
Tuscon, AZ 85721

1 Dr. Kenneth A. Klivington
Program Officer
Alfred P. Sloan Foundation
630 Fifth Avenue
New York, NY 10111

1 Dr. Stephen Kosslyn
Harvard University
Department of Psychology
33 Kirkland Street
Cambridge, MA 02138

1 Mr. Marlin Kroger
1117 Via Goleta
Palos Verdes Estates, CA 90274

1 Dr. Alan Lesgold
Learning R&D Center
University of Pittsburgh
Pittsburgh, PA 15260

1 Dr. Robert A. Levit
Director, Behavioral Sciences
The BDM Corporation
7915 Jones Branch Drive
McClean, VA 22101

1 Dr. Charles Lewis
Faculteit Sociale Wetenschappen
Rijksuniversiteit Groningen
Oude Boteringestraat
Groningen
NETHERLANDS

1 Dr. Erik McWilliams
Science Education Dev. and Research
National Science Foundation
Washington, DC 20550

1 Dr. Mark Miller
Computer Science Laboratory
Texas Instruments, Inc.
Mail Station 371, P.O. Box 225936
Dallas, TX 75265

1 Dr. Allen Munro
Behavioral Technology Laboratories
1845 Elena Ave., Fourth Floor
Redondo Beach, CA 90277

1 Dr. Donald A Norman
Dept. of Psychology C-009
Univ. of California, San Diego
La Jolla, CA 92093

1 Dr. Jesse Orlansky
Institute for Defense Analyses
400 Army Navy Drive
Arlington, VA 22202

1 Dr. Seymour A. Papert
Massachusetts Institute of Technology
Artificial Intelligence Lab
545 Technology Square
Cambridge, MA 02139

1 Dr. James A. Paulson
Portland State University
P.O. Box 751
Portland, OR 97207

1 MR. LUIGI PETRULLO
2431 N. EDGEWOOD STREET
ARLINGTON, VA 22207

1 Dr. Steven E. Poltrock
Department of Psychology
University of Denver
Denver,CO 80208

1 MINRAT M. L. RAUCH
P II 4
BUNDESMINISTERIUM DER VERTEIDIGUNG
POSTFACH 1328
D-53 BONN 1, GERMANY

1 Dr. Fred Reif
SESAME
c/o Physics Department
University of California
Berkely, CA 94720

1 Dr. Andrew M. Rose
American Institutes for Research
1055 Thomas Jefferson St. NW
Washington, DC 20007

1 Dr. Ernst Z. Rothkopf
Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

1 DR. WALTER SCHNEIDER
DEPT. OF PSYCHOLOGY
UNIVERSITY OF ILLINOIS
CHAMPAIGN, IL 61820

1 Dr. Alan Schoenfeld
Department of Mathematics
Hamilton College
Clinton, NY 13323

1 DR. ROBERT J. SEIDEL
INSTRUCTIONAL TECHNOLOGY GROUP
HUMRRO
300 N. WASHINGTON ST.
ALEXANDRIA, VA 22314

1 Committee on Cognitive Research
% Dr. Lonnie R. Sherrod
Social Science Research Council
605 Third Avenue
New York, NY 10016

1 Robert S. Siegler
Associate Professor
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

1 Dr. Edward E. Smith
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02138

1 Dr. Robert Smith
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903

1 Dr. Richard Snow
School of Education
Stanford University
Stanford, CA 94305

1 Dr. Robert Sternberg
Dept. of Psychology
Yale University
Box 11A, Yale Station
New Haven, CT 06520

1 DR. ALBERT STEVENS
BOLT BERANEK & NEWMAN, INC.
50 MOULTON STREET
CAMBRIDGE, MA 02138

1 David E. Stone, Ph.D.
Hazeltine Corporation
7680 Old Springhouse Road
McLean, VA 22102

1 DR. PATRICK SUPPES
INSTITUTE FOR MATHEMATICAL STUDIES IN
THE SOCIAL SCIENCES
STANFORD UNIVERSITY
STANFORD, CA 94305

1 Dr. Kikumi Tatsuoka
Computer Based Education Research
Laboratory
252 Engineering Research Laboratory
University of Illinois
Urbana, IL 61801

1 Dr. Douglas Towne
Univ. of So. California
Behavioral Technology Labs
1845 S. Elena Ave.
Redondo Beach, CA 90277

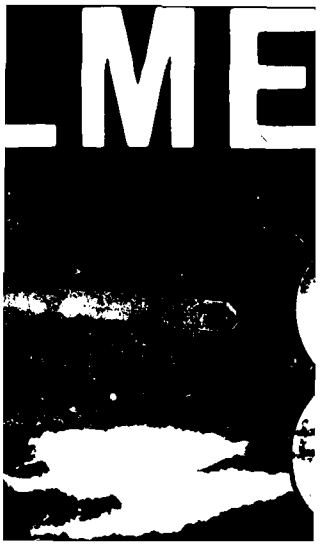
1 Dr. J. Uhlaner
Perceptronics, Inc.
6271 Variel Avenue
Woodland Hills, CA 91364

1 Dr. Phyllis Weaver
Graduate School of Education
Harvard University
200 Larsen Hall, Appian Way
Cambridge, MA 02138

1 Dr. David J. Weiss
N660 Elliott Hall
University of Minnesota
75 E. River Road
Minneapolis, MN 55455

1 DR. GERSHON WELTMAN
PERCEPTRONICS INC.
6271 VARIEL AVE.
WOODLAND HILLS, CA 91367

1 Dr. Keith T. Wescourt
Information Sciences Dept.
The Rand Corporation
1700 Main St.
Santa Monica, CA 90406


LME