PROUST: Knowledge-Based Program Understanding
W. Lewis Johnson and Elliot Soloway
YaleU/CSD/RR #235
August 1983

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited
PROUST: Knowledge-Based Program Understanding
W. Lewis Johnson and Elliot Soloway
YaleU/CSD/RR #235
August 1983
# Expert Systems Programming Plans

## ABSTRACT

This paper describes a program called PROUST which does on-line analysis and understanding of Pascal programs written by novice programmers. PROUST takes as input a program and a non-algorithmic description of the program requirements, and finds the most likely mapping between the requirements and the code. This mapping is in essence a reconstruction of the design and implementation steps that the programmer went through in writing the program. A knowledge base of programming plans and strategies, together with common bugs associated with them, is used in constructing this mapping. Bugs are discovered in the process of...
relating plans to the code; PROUST can therefore give deep explanations of program bugs by relating the buggy code to its underlying intentions.
PROUST: Knowledge-Based Program Understanding

W. Lewis Johnson
Elliot Soloway

July 1983

Yale University
Computer Science Department
New Haven, Ct. 06520
203-438-0606

This work was co-sponsored by the Personnel and Training Research Groups, Psychological Sciences Division, Office of Naval Research and the Army Research Institute for the Behavioral and Social Sciences, under Contract No. N00014-82-K-0714, Contract Authority Identification Number, Nr 154-492. Approved for public release; distribution unlimited. Reproduction in whole or part is permitted for any purpose of the United States Government.
Abstract

This paper describes a program called PROUST which does on-line analysis and understanding of Pascal programs written by novice programmers. PROUST takes as input a program and a non-algorithmic description of the program requirements, and finds the the most likely mapping between the requirements and the code. This mapping is in essence a reconstruction of the design and implementation steps that the programmer went through in writing the program. A knowledge base of programming plans and strategies, together with common bugs associated with them, is used in constructing this mapping. Bugs are discovered in the process of relating plans to the code; PROUST can therefore give deep explanations of program bugs by relating the buggy code to its underlying intentions.

1. Introduction: Motivation and Goals

Our goal is to build a tutoring system which helps novice programmers to learn how to program. This system will have two components: a programming expert which can analyze and understand buggy programs, and a pedagogical expert that knows how to effectively interact with and instruct students. We have focused our attention on the first component, with the objective of building a system that can be said to truly understand (buggy) novice programs. In this paper, we will describe the theory and processing techniques by which our analysis system, PROUST, understands buggy and correct programs.

Bugs in programs are sections of code whose behavior fails to agree with the program specification. Although the presence of bugs may be indicated by various kinds of anomalous program behavior, in general bugs are not properties of programs, but rather are properties of the relationship between programs and intentions. [9, 10] For example, consider the program in Figure 1-1. The programmer has written a program that reads in a number and then computes the average of all the numbers between it and 99999, in integer increments. This is not what the stated problem requires; presumably the programmer was trying to solve the problem, but a bug has altered the program’s behavior. How do we determine what this bug is? Note that the programmer first does a Read into the variable New, and then increments it by 1. Based on our theory of programming knowledge, [17, 12, 18, 1] we would hypothesize that the student thought that incrementing the variable New would return the next value of New; if incrementing Count gets the next INTEGER value, then incrementing New should get the next input value! The student has thus made an overgeneralization: adding one to a variable returns the next value of that variable. The key element of the above analysis is the construction of a relationship from a piece of code to a problem goal; the mechanism for that construction was knowledge about how

1Miller’s SPACE-o [11] is another example of a programming tutor; unlike PROUST, it constrains the program construction process so that less machinery is required for understanding and more effort can be devoted to pedagogy.
programs are typically constructed, together with knowledge about novice misconceptions.

Problem: Read in numbers, taking their sum, until the number 99999 is seen. Report the average. Do not include the final 99999 in the average.

```plaintext
1 PROGRAM Average( input, output );
2 VAR Sum, Count, New, Avg: REAL;
3 BEGIN
4 Sum := 0;
5 Count := 0;
6 Read( New );
7 WHILE New<>99999 DO
8 BEGIN
9 Sum := Sum+New;
10 Count := Count+1;
11 New := New+1
12 END;
13 Avg := Sum/Count;
14 Writeln( 'The average is', avg );
15 END;
```

PROUST output:

It appears that you were trying to use line 11 to read the next input value. Incrementing NEW will not cause the next value to be read in. You need to use a READ statement here, such as you use in line 6.

Figure 1-1: Example of analysis of a buggy program

While we have not built a pedagogical expert yet, it would certainly need the type of information produced in the above analysis. That is, an intelligent tutoring system would need to know:

- what the bugs in the student's program are, and where they occur;
- what the student was intending to do with the buggy code;
- what misconceptions the student might have which would explain the presence of the bugs.

What is an appropriate method for deriving information such as this from a program? One way might be to compare the input-output behavior of the program against the expected input-output behavior. The information which this approach would provide is insufficient, particularly with larger programs, because a number of bugs might result in the same input/output behavior.²

²BIP [21] makes use of input/output behavior in its program analysis; consequently it only deals with small programming problems.
For example, many different bugs can cause a program to go into an infinite loop, so simply knowing that a program goes into an infinite loop is insufficient for determining what the bug is. Enhancing input-output analysis with dataflow analysis, or other compiler analysis techniques, will not help in cases where the code does not have any obvious structural anomalies,\(^3\) such as in the preceding example.

What is missing in the above methods is a detailed understanding of the relationship between the program text and the program's intentions. We suggest that a method for building such a description involves (1) recreating the goals that the student was attempting to solve (i.e., what problem the student thought he was solving), (2) identifying the functional units in the program that were intended to realize those goals. In effect, the programming expert needs to analyze the buggy program by reconstructing the manner in which it was generated. The claim is that the trace generated by the programming expert does actually correspond to what the student was thinking, although not necessarily to the utmost detail; the pedagogical expert would then use that trace in subsequent tutoring activity.\(^4\) In this paper, we briefly highlight the theoretical basis for reconstructive program analysis, and we detail how PROUST goes about building the reconstruction.

2. The Role of Plans in Program Understanding

Knowledge about what implementation methods should be used in programming is codified in PROUST in the form of programming plans. A programming plan is a procedure or strategy for realizing intentions in code, where the key elements have been abstracted and represented explicitly. It is our position that expert programmers make extensive use of programming plans, rather than each time building programs out of the primitive constructs of a programming language. This claim is based on a theory of what mental representations programmers have and use in reading and writing programs. In [17, 6, 19, 20] we describe various empirical experiments which support our theory. Thus, PROUST is directly based on a plausible, psychological theory of the programming process. Note that codifying programming knowledge in terms of plans is not unique to PROUST; the Programmer's Apprentice, [12] for example, also makes extensive use of plans.\(^5\)

Figure 2-1 is an illustration of how plans are realized in programs. The figure shows a correct

---

\(^3\) One area in which many compilers do a reasonable job is analyzing syntactic errors. Although it would be worthwhile to construct a parser which produces error reports aimed at novices, this is outside of the scope of our current work.

\(^4\) Most intelligent tutoring systems at least tacitly assume such a correspondence. [7, 8, 31]

\(^5\) Sniffer [15] is a prototype of a debugging system which is based upon the Programmer's Apprentice.
implementation of the problem shown in Figure 1-1, together with four plans that this program uses. Two of them, the RUNNING TOTAL VARIABLE PLAN and the COUNTER VARIABLE PLAN, are variable plans, i.e. they are plans which generate a result which is usually stored in a variable. Such plans typically have an initialization section and an update section, and carry information about what context they must appear in, e.g. whether or not they must be enclosed in a loop. The other two plans, the RUNNING TOTAL LOOP PLAN and the VALID RESULT SKIP GUARD, are control plans; their main role is not to generate results but to regulate the generation and use of data by other plans. The RUNNING TOTAL LOOP PLAN is a method for constructing a loop which controls the computation of a running total; in this program it also controls the operation of the COUNTER VARIABLE PLAN. The VALID RESULT SKIP GUARD plan is an example of a skip guard, i.e. a control plan which causes control flow to skip around other code when boundary conditions occur. In this case it prevents the average from being computed or output when there is no input.

Problem: Read in numbers, taking their sum, until the number 99999 is seen. Report the average. Do not include the final 99999 in the average.

PROGRAM Average (INPUT, OUTPUT);
VAR Sum, Count, New, Avg: REAL;
BEGIN
Plan
---------- > Count := 0;
| --- > Sum := 0;
| | Read(New); <---------------
| | Running Total Loop Plan
| | WHILE New <> 99999 DO -------|
| | Variable Plan
| | BEGIN
| | | --- > Sum := Sum + New; <------
| | | --------- > Count := Count + 1;
| | | Read(New); <---------
| | END;
| | Valid Result Skip Guard
| IF Count > 0 THEN <-----------------------------
| BEGIN <----------------------------------------
| Avg := Sum/Count; <--------------------------------
| WriteIn( Avg); <--------------------------------
| END <----------------------------------------
| ELSE <----------------------------------------
| WriteIn( 'no legal inputs'); <----------------
| END.

Figure 2-1: Programming Plans

Recognition of plans in programs forms the basis of our approach to program understanding. But plan recognition alone is insufficient. Novices often use plans that would never occur to an expert, because they do not have a good sense of what is a good plan and what is not. PROUDT's knowledge base of plans has therefore been extended in order to include many stylistically dubious plans. Unfortunately, the more alternative plans there are in the system, the harder it

---

6 The process of collecting novice programs and analyzing them is described in [2], [9], and [10].
is to determine which plans the programmer was using. Furthermore, program behavior depends not only upon what plans are used, but how they are organized; it is thus possible for a program to use correct plans yet still have bugs. In order to cope with these problems a method is needed for relating plans to other plans, and to the programmer's underlying intentions. This process, and the way it is used to search for the right interpretation of the program, is described in Section 4.

3. A Typical Problem in PROUST's Domain

PROUST's knowledge base is currently tailored to analyze the programming problem in Figure 3-1. This problem (hereafter referred to as the Rainfall Problem) is a more complex version of the averaging problem shown in Figure 1-1. Among other computations, a program that solves this problem must

1. count the number of valid inputs (i.e., days on which there was zero or greater rainfall), and
2. count the number of positive inputs (i.e., days on which rain fell).

Novices attempt to realize these two goals in a variety of correct and buggy ways. Since coping with variability is one of PROUST's main objectives, examining how PROUST handles this specific set of goals should be illustrative. Thus, in what follows, we will focus on PROUST's techniques for processing fragments of code that implement these goals.

Noah needs to keep track of rainfall in the New Haven area in order to determine when to launch his ark. Write a program which he can use to do this. Your program should read the rainfall for each day, stopping when Noah types "9999", which is not a data value, but a sentinel indicating the end of input. If the user types in a negative value the program should reject it, since negative rainfall is not possible. Your program should print out the number of valid days typed in, the number of rainy days, the average rainfall per day over the period, and the maximum amount of rainfall that fell on any one day.

Figure 3-1: The Rainfall Problem

4. Relating Goals to Code via Plans

In order to relate the plans in a program to the program requirements, PROUST makes explicit the goal decomposition underlying the program. A goal decomposition consists of

- a description of the hierarchical organization of the subtasks in a problem,
- indications of the relationships and interactions among subtasks, and
- a mapping from subtask requirements (goals) to the plans that are used to implement them.

---

*We are currently extending PROUST to handle a range of introductory programming problems.*
The plans which a goal decomposition specifies are matched against the program; this results in a mapping from program requirements to individual statements.

In attempting to understand all except the most trivial programming problems, two issues must be squarely faced:

- the goal decomposition of a problem may not be unique, and
- one program may be associated with more than one goal decomposition.

We deal with each issue in turn in the next two sections.

### 4.1. The Space of Goal Decompositions and Programs

Figure 4-1 illustrates how alternative goal decompositions can lead to different program implementations. A single problem description, at the top, can result in several different goal decompositions, which in turn result in a number of different programs, depending upon which plans are used. Some of these programs may be correct, others buggy. Buggy programs are either derived from incorrect goal decompositions or from incorrect implementations of correct goal decompositions. Each path from the problem description down to an individual program is a program interpretation; we call this set of possible derivation paths the interpretation space associated with a problem.

![Figure 4-1: Search space of possible programs](image)

Figures 4-2 and 4-3 illustrate two different solutions of the Rainfall Problem (Figure 3-1) and their corresponding goal decompositions. We focus here on two specific aspects of the problem:  

1. counting the valid inputs (daily rainfall greater than or equal to zero), and
2. counting the number of rainy days (daily rainfall strictly greater than zero).

Figure 4-2 shows a fragment in which these two goals are realized directly. First, a COUNTER VARIABLE PLAN is used to count the valid inputs; this is realized in the code that computes the

---

There are other differences in the goal decompositions of these programs besides the ones mentioned here. However, we will not analyze them in this discussion.
value of the variable \texttt{Valid}. Second, the \texttt{GUARDED COUNTER VARIABLE PLAN} is used for counting the positive inputs; the variable \texttt{Rainy} is used in this plan.

While the program in Figure 4-3 also prints out the number of valid inputs and the number of positive inputs, the goal decomposition in this program is different. Instead of the two goals of counting the valid inputs and counting the positive inputs, the program in Figure 4-3 uses three goals to achieve the same end: (1) count the zero inputs, (2) count the positive inputs, and (3) add these two counters together to derive the valid day total. The goal of counting the positive inputs is implemented with a \texttt{GUARDED COUNTER VARIABLE PLAN}, operating on the variable \texttt{Rainy}. The goal of counting the zero inputs is also implemented with a \texttt{GUARDED COUNTER VARIABLE PLAN}, operating on the variable \texttt{Dry}. The counters are combined with an \texttt{ADD PARTIAL RESULTS PLAN}, resulting in the variable \texttt{Valid}.

4.2. Resolving Ambiguous Interpretations

If the mapping from problem descriptions to programs is to be rich enough to generate a sufficiently wide variety of programs, ambiguity is an unavoidable consequence, i.e. two different paths in the interpretation space can lead to the same program. This situation is exacerbated when buggy programs are allowed: bugs add uncertainty to the analysis. For example, if one encounters a statement \texttt{New := New+1} in a correct program, one can be fairly certain that it is part of a counter plan. But if the program is buggy, as in Figure 1-1, one must also consider the possibility that this statement is intended to input new values; the only way of determining which is the proper role is by looking at the program as a whole and determining which interpretation is more consistent with the interpretations of the other parts of the program. The ability to enumerate and evaluate alternative interpretations is a key processing technique for a system that attempts to understand buggy programs.

In Figure 4-4 we give an example of the results of PROUST’s attempt to resolve ambiguous interpretations. Figure 4-4 shows a fragment of code which might appear in a novice solution to the Rainfall Problem in Figure 3-1. We have focused on the counter variables in the program, \texttt{Valid} and \texttt{Rainy}; the rest of the main loop of the program is shown so that the surrounding context may be seen. Instead of counting the positive inputs (\texttt{Rain>0}) and the valid inputs (\texttt{Rain>=0}), this program counts the positive inputs and the zero inputs, and does not count the valid inputs.

There are two possible interpretations for this code, each of which results in a different explanation for the bugs. According to one interpretation, shown on the left side of the figure, the programmer intended to implement the valid input goal and the positive input goal directly. The plans used are \texttt{COUNTER VARIABLE PLAN} and \texttt{GUARDED COUNTER VARIABLE PLAN}; the resulting variables are \texttt{Valid} and \texttt{Rainy}, respectively. \texttt{Valid} appears to count only the the zero
Plans

Plan: RUNNING TOTAL LOOP PLAN

Plan: COUNTER VARIABLE PLAN

Plan: GUARDED COUNTER VARIABLE PLAN

Goal Decomposition

1. Get input, stopping at 99999
2. Check that input is non-negative...
6. Count valid inputs
7. Count positive inputs...

PROGRAM Rainl (INPUT, OUTPUT):
CONST STOP=99999:
VAR
  Sum, Rain, Max, Ave: REAL;
  Valid, Rainy: INTEGER;
BEGIN
  Writeln('Enter rainfall');
  Sum:=0;
  Valid:=0;
  Rainy:=0;
  Max:=0;
  Readln;
  Read(Rain);
  WHILE Rain<>STOP OR
  BEGIN
    IF Rain<0 THEN
      Writeln(Rain:0.2, ' is not a possible rainfall. Try again')
    ELSE
      BEGIN
        Sum:=Sum+Rain;
        Valid:=Valid+1;
        IF Rain>Max THEN
          Max:=Rain;
        IF Rain>0 THEN
          Rainy:=Rainy+1;
        END;
        Writeln('Enter rainfall');
        Readln;
        Read(Rain)
      END;
    Writeln.
    Writeln(Valid:0, ' valid rainfalls were entered.');
    IF Valid>0 THEN
      BEGIN
        Ave:=Sum/Valid;
        Writeln('The average rainfall was ', Ave:0.2, ' inches PER DAY. ');
        Writeln('The highest rainfall was ', Max:0.2, ' Inches. ');
        Writeln('There were ', Rainy:0, ' rainy days in this period. ')
      END
    END.
END.

Figure 4-2: Simple goal decomposition
Plans

Plan: RUNNING TOTAL LOOP PLAN

Plan: COUNTER VARIABLE PLAN

Plan: GUARDED COUNTER VARIABLE PLAN

Goal Decomposition

1. Get input, stopping at 9999
2. Check that input is non-negative
6. Count zero inputs
7. Count positive inputs
8. Combine counters

PROGRAM Rain2 (INPUT, OUTPUT):
CONST STOP=9999;
VAR Sum,Rain,Max,Ave: REAL;
   Valid,Rainy,Dry: INTEGER;
BEGIN
   Sum:=0;
   Dry:=0;
   Rainy:=0;
   Max:=0;
   Writeln('Enter rainfall');
   Readln;
   Read(Rain);
   WHILE Rain<0 DO
      BEGIN
         Writeln(Rain:0:2,' is not a possible rainfall, try again');
         Readln(Rain);
      END;
   WHILE Rain<>STOP DO
   BEGIN
      Sum:=Sum+Rain;
      IF Rain=0 THEN
         Dry:=Dry+1
      ELSE
         Rainy:=Rainy+1;
      IF Rain>Max THEN Max:=Rain;
      Valid:=Rainy+Dry;
      Writeln('Enter rainfall');
      Readln;
      Read(Rain);
   WHILE Rain<0 DO
      BEGIN
         Writeln(Rain:0:2,' is not a possible rainfall, try again');
         Readln(Rain);
      END;
   END;
   Writeln;
   Writeln(Valid:0,' valid rainfalls were entered.');
   IF Valid>0 THEN
      BEGIN
         Ave:=Sum/Valid;
         Writeln('The average rainfall was ',Ave:0:2,' inches per day.');
         Writeln('The highest rainfall was ',Max:0:2,' inches.');
         Writeln('There were ',Rainy:0,' rainy days in this period.');
      END;
   END.

Figure 4-3: Alternative goal decomposition
Buggy Program Fragment

```
WHILE Rain<99999 DO
BEGIN
  IF Rain=0 THEN
    Writeln('Input not valid')
  ELSE
    BEGIN
      IF Rain=0 THEN
        Valid := Valid+1
      ELSE
        BEGIN
          Rainy := Rainy+1;
          Sum := Sum+Rain.
        END;
    END;
  END;
END;
Writeln('Enter next value').
Read(Rain).
END.
```

Goal Decomposition 1

Goal: count all items
Goal: count positive items

Bug: Missing copy of duplicated plan segment

Explanation to student:
This program will not count the number of inputs correctly.
You increment "Valid" when the input is zero, but not when it is positive.

Goal Decomposition 2

Goal: count zero
Goal: count positives
Goal: combine partial counts

Bug: Mushed variables
Missing plan

Explanation to student:
You are using the variable "Valid" both to count the total number of inputs and the number of zero inputs. Each variable should be used to mean one and only one thing. Also, you are going to have to add the zero count and the positive count together.

Figure 4-4: Alternative explanations for bugs

inputs, because the programmer intended to modify the COUNTER VARIABLE PLAN so that a copy of the counter update appears in both the THEN branch and the ELSE branch of the inner IF statement, and then left out one of the copies. The failure to update Valid in both branches thus appears to be a low-level slip, such as a mistake in editing the source file.

In the other interpretation, on the right side of the of the figure, the program is assumed to arise from a goal decomposition where the positive values and the zero values are counted separately and then added together. The programmer uses the variable Valid to refer to the count of zero values and Rainy to refer to the count of positive values. The plan to add Valid and Rainy together is missing. We could claim that the plan is missing because of an editing slip.
However, the context in which the counter plans appear weighs against this hypothesis: the
average computation uses Valid in the denominator of the division, implying that Valid is the
valid input counter as well as the zero input counter. We call variables which are used in
contradictory ways such as this mushed variables. Mushed variables are very serious bugs; they
reflect radical deficiencies in the programmer's ability to design programs. Therefore this goal
decomposition is less highly valued than the previous goal decomposition. PROUST has a number
of heuristics for deciding among alternative interpretations such as these.

5. The Understanding Process: An Example Of PROUST In Action
In the preceding sections, we (1) described what difficulties a program understanding system
must overcome in order to analyze a program accurately, and we (2) gave an example of the
results of PROUST's analysis. In this section, we will illustrate PROUST's processing capabilities.
First we will describe the overall strategy by which PROUST searches through the space of
potential interpretations for one that best accounts for the student's program, and then we will
describe how PROUST actually produces the analysis already depicted in Figure 4-4.

5.1. Searching the Interpretation Space
Clearly, one can't possibly enumerate beforehand the space of program interpretations: there
are just too many ways to construct correct and buggy programs. Rather, starting with the
problem specification and a database of correct and buggy plans, transformation rules\(^9\), and bug-
misconception rules, PROUST constructs and evaluates interpretations for the program under
consideration. In effect, the goal decomposition and the plan analysis of the program evolve
simultaneously. To constrain the generation process, PROUST employs heuristics about what
plans and goals are likely to occur together.

The evaluation process is prediction driven: based on the current candidate interpretation for
the program, how well do other parts of the program conform to PROUST's expectations? For
example, if, in a program that attempts to solve the Rainfall Problem, PROUST has assumed that
the variable Count is keeping track of the number of valid days, PROUST would expect to see
Count in the denominator of the average daily rainfall calculation. If this expectation is
confirmed, then PROUST is more confident of its interpretation, and vice versa. PROUST employs
heuristics that evaluate matches, near-misses, and misses of its expectations. Examples of
construction and evaluation processes will be given in the next section.

The fact that PROUST constructs and evaluates interpretations anew for each program, and

\(^9\)These entities will be explained shortly.
does not rely on a prestored set of possible interpretations, provides it with an important capability: PROUST readily generates interpretations for programs that it (and we) have not seen previously. That is, unlike some diagnostic systems that effectively choose a fault from a set of predefined faults, [16, 4] PROUST actively constructs diagnoses. Given the variability in programs, PROUST needs such a capability in order to be effective.  

5.2. Putting It All Together: Two Examples

```plaintext
Sum := 0;
Rainy := 0;
Valid := 0;
Max := 0;
Read( Rain);
WHILE Rain<>99999 DO
BEGIN
  IF Rain<0 THEN
    Writeln( 'Input not valid' )
  ELSE
    BEGIN
      IF Rain=0 THEN (a)
      Valid := Valid+1 (b)
      ELSE
        BEGIN
          Valid := Valid+1; (c)
          Rainy := Rainy+1;
        END;
      END;
    END;
  Sum := Sum+Rain;
  IF Rain>Max THEN
    Max := Rain;
  END;
  Writeln( 'Enter next value:' );
  Read( Rain );
END;
Avg := Sum/Valid;
```

Figure 5-1: Excerpt of Rainfall Program

In this section we will illustrate how PROUST actually goes about analyzing a program. We will show two examples; one is a correct program and the other is a buggy program.

5.2.1. Analysis of a correct program

Our first example, in Figure 5-1, is an excerpt from a correct solution to the Rainfall Problem in Figure 3-1; it is based on the program fragment shown in Figure 4-4. Although this program functions correctly, there is one construction which is unusual; the valid input counter `Valid` is updated in two places instead of one. That is, `Valid` is updated in each branch of the conditional

14) PALOSY [14] is also capable for recognizing novel faults; however, it assumes that there is only one fault, which the programmer must describe beforehand.
statement at (a); the update at (b) is executed when Rain is zero, and the update at (c) when Rain is positive. The program in this figure illustrates the variability possible in programs: coping with this type of situation requires additional machinery, as will be seen shortly.

Assume that PROUST has carried out a partial plan analysis of this program already, and has made the following tentative assumptions:

- the variable Sum is the running total variable,
- the variable Valid keeps tracks of the number of valid days,
- the update on Valid should be in the loop, embedded inside a test for negative rainfall (IF Rain < 0 THEN ...).

The processing that continues from this point is illustrated in Figure 5-2. PROUST maintains an agenda of goals that remain to be worked on; at this point in the analysis the agenda includes the Count goal for valid inputs, the Sum goal, and the Count goal for positive inputs, to name a few. PROUST selects the first goal on the agenda, as shown at (a), checks that it is ready for analysis, and then determines whether or not it needs to be decomposed. The entry in the knowledge base for Count stipulates that it is most commonly implemented in an undecomposed fashion, so Proust consults the plan database looking for appropriate plans for realizing this goal. It finds only one plan plan: the COUNTER VARIABLE PLAN (b). It then makes tentative bindings for the plan variables, and determines where each segment of the plan should be found. The resulting structure, shown at (c), can then be matched against the student's program.

Figure 5-3 shows the results of matching the instantiated plan against the code. There is a unique match for the initialization step of the plan, but instead of there being one match for the update step, there are two matches. Furthermore, PROUST expects the update to be at "top level" inside the loop, i.e. it should not be enclosed inside code which might disrupt its function. Instead it discovers that each update is enclosed in an IF statement which restricts its application. PROUST treats the plan as a near-match for the program, but the plan cannot be accepted until the match discrepancies are accounted for.

PROUST has a number of different methods for explaining a plan difference; one of them is to use transformation rules to relate the code to the plan. One such transformation is shown in Figure 5-4.11 Each transformation rule has a test part and an action part. The test part consists of a conjunction of micro-tests, each testing various aspects of the program; the action part usually indicates how to modify the code in order to nullify the effect of the transformation. In this case the Distribution Transformation Rule applies. This is a rule for recognizing plans in situations where a set of computations have been divided into parts using a CASE statement or an

---

11PROUST currently has 15 such transformations in its database. Some rules, such as the Distribution Transformation Rule, are quite general; others, such as the transformation which changes Valid<>0 into Valid<>0 IF Valid is a counter variable, are plan specific.
Agenda of Goals

goal: Count( fNew=Rain, fCount=Valid)
goal: Sum( fNew=Rain, fTotal=Sun)
goal: Count( fNew=Rain, ai. fNew>0)

a) goal selection

goal: Count(?Count=Valid)

b) plan retrieval

counter variable plan

init step: ?Count = 0
update step: ?Count = ?Count+1

c) tentative variable bindings

?Count=Valid

counter variable plan

init step: Valid = 0
(above WHILE loop)
update step: Valid = Valid+1
(in WHILE loop above Read)

Figure 5-2: Simple mapping from goals to instantiated plans

IF-THEN-ELSE construct, and where the plan update is duplicated so that a copy appears in each branch. The control flow branches in this case are the two branches in the IF-THEN-ELSE construction which test for Rain=0 and Rain>0. The rule checks to see whether there is exactly one Valid:Valid+1 statement for each possible branch of the test. It then combines the two updates and moves the result to an appropriate place outside of the test. Once this is done the counter plan matches successfully.

5.2.2. A buggy example

We will now show how PROUST analyzes the buggy program shown in Figure 4-4: a more complete version is given in Figure 5-5. When PROUST analyzes buggy programs such as this, it goes through much the same process that it goes through in analyzing correct programs: the main difference is that PROUST must consider more alternative interpretations in order to find the most
Sum := 0; Valid := 0; Rainy := 0; Max := 0; Read( Rain ); WHILE Rain<>99999 DO BEGIN IF Rain<0 THEN Writeln( 'Input not valid' ) ELSE BEGIN IF Rain=0 THEN Valid := Valid+1 Valid := Valid+1 ELSE BEGIN Valid := Valid+1 Valid := Valid+1 Rainy := Rainy+1 Rainy := Rainy+1 END; END; IF Rain>Max THEN Max := Rain; END; Writeln( 'Enter next value:' ); Read( Rain ); END; Avg := Sum/Valid;

**Figure 5-3:** Plan matching

**Figure 5-4:** Program transformation

plausible explanation for the bug.

Figure 5-6 shows what happens when the COUNTER VARIABLE PLAN is matched against this program. This time there is one good match for the counter update; unfortunately it is inside of an unexpected IF statement. The Distribution Transformation Rule is invoked to explain the plan difference, but it predicts that there should be two updates, so it does not fully explain the problem. PROUST therefore looks for another rule which will explain the difference between the prediction made by the Distribution Transformation Rule and the observed code. A rule applies which states that if an single instance of duplicated code is missing, it is explainable as a low-level slip. This completes the mapping from the plan to the code.

Whenever an interpretation presumes the presence of a bug, it is necessary to make sure that
there are no other interpretations which presume fewer or less severe bugs. PROUST therefore goes back and looks for another way of implementing the Count goal. PROUST has in its knowledge base an alternative method for decomposing Count goals, namely to implement counters for particular intervals and then combine the partial counts. One of these subgoals can be unified with the Count positives goal that already exists in the agenda. The two Count goals are thus transformed into a set of three goals. Plans can then be chosen and instantiated for each of these goals, as was done in Figure 5-2. The result plans, and the results of matching them, is shown in Figure 5-7. This time two match errors are found. First, Valid is the counter for zero values; but the average predicts that Valid is the main counter; Valid is a mushed variable. Second, the ADD PARTIAL RESULTS PLAN is missing altogether. PROUST ranks bugs according to their severity; missing plans that do not pertain to some boundary condition are moderately severe bugs, and mushed variables are extremely severe bugs. Therefore this interpretation is less highly valued, and the analysis involving the transformation holds.
6. Performance – Preliminary Results

As a preliminary test of PROUST's capabilities, we tested PROUST on 206 different novice solutions to the Rainfall Problem shown above. We collected these programs by modifying the Pascal compiler used by the students in an introductory programming course so that each syntactically correct version of the program was stored on tape [2]. We ran PROUST on the first syntactically correct version from each student, so that we could see how PROUST behaves when faced with a large number of bugs.

In Table 6-1 we see how PROUST performed on this corpus of programs. Of the 206 programs in the sample, PROUST only commented on 137 of them (67%). The remaining 33% PROUST decided that it didn't understand the program well enough to make a reasonable assessment of the bugs. Thus, rather than venturing a guess, PROUST remained silent. On those programs that it did feel confident of its analysis, it was correct almost 94% of the time! In an educational setting, we felt that no advice was better than bad advice. Thus, we built into

---

There were still 32 "false alarms:" cases where PROUST said there was a bug, but there really wasn't.
Figure 5-7: Matching alternate plans

PROUST a number of heuristics that it would use to assess its confidence in its analysis. From the data in Table 6-1, it seems that when PROUST thought it had a good analysis, it was indeed correct.

Total number of programs: 206

PROUST actually gave complete bug reports for 137 programs (67%)

Total number of bugs (from 137 programs) 444
- Bugs recognized correctly: 419 (94%)
- Bugs not reported: 25 (6%)
- False alarms: 32

Table 6-1: Preliminary results

Clearly, the next stage is to improve PROUST's overall performance. Moreover, in looking at the cases where PROUST failed, we see no fundamental obstacle to getting PROUST up to the 80% overall correct rate. However, we can characterize the kinds of programs which will always cause problems for PROUST as follows: 1) very unusual bugs, which occur too infrequently to justify inclusion in PROUST's knowledge base, 2) programs which contain novel plans which PROUST has no means for predicting, 3) ambiguous cases which can only be resolved through dialog with the student. For these cases, we would suggest that the student see...
7. Concluding Remarks

Is all the machinery described in this paper necessary in order to understand buggy and correct programs -- programs that are only about 1 page in length? The answer, in our minds at least, is: undeniably yes. If anything, PROUST is the minimum that is required! The basis for this conclusion is twofold:

1. In Artificial Intelligence research, systems have been built to understand stories of moderate length that require machinery similar to that employed by PROUST. [13, 5] Certainly, programs are as complicated an entity as are stories.

2. We attempted to build a bug finding system that used a database of bug templates in a context-independent fashion to analyze programs similar to those analyzed by PROUST. That system, MENO, [18] failed miserably: in order to cope with the variety and variability in actual programs, a system must be able to understand how the pieces of the program fit together -- which is a highly context-dependent process.

Finally, all programmers intuitively know that the mapping from problem specifications to code is a complex process. What PROUST has done -- which we feel is its major contribution -- is lay that mapping process open to inspection: since PROUST constructs a program in its attempt to understand the program under analysis, we can "see" the programming process in action. By making the programming process explicit, our work joins with that of the software engineering community to change programming from an ethereal art to an object of scientific inquiry.
I. Proust's analysis of a sample program

```plaintext
PROGRAM RAINFALL (INPUT, OUTPUT):

CONST
SENTINEL = 99999;

VAR
RAINFALL, VALID, HIGHEST, AVERAGE, TOTAL : REAL;
RAINDAY : INTEGER;

BEGIN
(*INFORMATION IS ENTERED INTO THE TERMINAL*)
WRITELN('PLEASE ENTER THE AMOUNT OF RAINFALL FOR EACH DAY SEPARATELY');
WRITELN('THIS PROGRAM WILL THEN FIGURE OUT THE AVERAGE, HIGHEST, TOTAL');
WRITELN('NUMBER OF RAINY DAYS, AND THE NUMBER OF VALID RAINY DAYS ENTERED');
WRITELN('PLEASE MAKE SURE THE NUMBERS ARE POSITIVE');
WRITELN('ENTER RAINFALL');
READLN;
READ(RAINFALL);

(*TEST FOR INVALID ENTRY*)
IF RAINFALL <= 0 THEN
WRITELN('THE DATA IS IMPOSSIBLE PLEASE CHECK AND REENTER DATA');
READLN;
READ(RAINFALL);

(*IDENTIFICATION AND CALCULATIONS*)
BEGIN
WHILE RAINFALL <> SENTINEL DO
IF RAINFALL > HIGHEST THEN
HIGHEST := RAINFALL;
TOTAL := RAINFALL + TOTAL;
IF RAINFALL >= 0 THEN
VALID := VALID + 1;
IF RAINFALL > 0 THEN
RAINDAY := RAINDAY + 1;
END;
REPEAT
UNTIL RAINFALL = SENTINEL;
(*FIND AVERAGE*)
AVERAGE := (TOTAL / VALID);
(*PRINTS OUTPUT*)
WRITELN(' '.VALID:2.'VALID RAINFALLS WERE ENTERED.');
WRITELN('THE AVERAGE RAINFALL WAS'.AVERAGE:8:2.'INCHES PER DAY');
WRITELN('THE HIGHEST RAINFALL WAS'.HIGHEST:8:2.'INCHES');
WRITELN('THERE WERE'.RAINDAY:2.'RAINY DAYS IN THIS PERIOD')
END.
```
Bug Report

1. The WHILE statement at line 29 and the BEGIN statement at line 29 are reversed. The BEGIN should come after the WHILE.

2. Your main loop is missing a READ statement. As it stands your loop will process the same input value over and over. Perhaps you intended line 24 to serve this purpose. If so, there may be a bug there.

3. You left out the initialization for the variable VALID.

4. You left out the initialization for the variable TOTAL.

5. You left out the initialization for the variable RAINDAY.

6. You left out the initialization for the variable HIGHEST.

7. You need a test to check that at least one valid data point has been input before line 42 is executed. The average is not defined when there is no input.

8. You need a test to check that at least one valid data point has been input before line 46 is executed. The average is not defined when there is no input.

9. You need a test to check that at least one valid data point has been input before line 47 is executed. The maximum is not defined when there is no input. Perhaps you intended line 33 to serve this purpose. If so, there may be a bug there.

10. Your test for valid input at line 20 won't work, because it's outside the main loop. Remember that you have to test ALL the input for validity.

11. The loop at line 39 doesn't do anything; it will loop forever. In particular, it will not make the program loop back to the beginning, if that is what you had in mind.
References

[1] Bonar, J. and Soloway, E.
Uncovering Principles of Novice Programming.
1983.
SIGPLAN-SIGACT Tenth Symposium on the Principles of Programming Languages, in press.

Collecting and Analyzing On-Line Protocols from Novice Programmers.

Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPHIE I, II, and III.

Applications-oriented AI Research: Education.

[5] Dyer, M.
*In-Depth Understanding*.
Technical Report 219, Computer Science Department, Yale University, May, 1982.

An Empirical Investigation of the Tacit Plan Knowledge in Programming.
1983.

The Role of Plans in Intelligent Teaching Systems.

[8] Goldstein, I. P.
The Genetic Graph: a Representation for the Evolution of Procedural Knowledge.

An Effective Bug Classification Scheme Must Take the Programmer into Account.
1983.
SIGPLAN/SIGSOFT Workshop on High-Level Debugging, in press.

Classifying Bugs is a Tricky Business.
1983.
NASA Workshop on Software Engineering, in press.
A Structured Planning and Debugging Environment for Elementary Programming.

A Formal Representation for Plans in the Programmer's Apprentice.
In Proc. of the Seventh Int. Joint Conf. on Artificial Intelligence, pages 1044-1052.
ICJAI, August, 1981.

Scripts, Plans, Goals, and Understanding.

[14] Sedlmeyer, R. L. and Johnson, P. E.
Diagnostic Reasoning in Software Fault Localization.
In Proceedings of the SIGSOFT Workshop on High-Level Debugging. SIGSOFT,
Asilomar, Calif., 1983.

[15] Shapiro, D. G.
Sniffer: a System that Understands Bugs.

[16] Soloway, E., Rubin, E., Woolf, B., and Bonar, J.
MENO-II: An AI-CAI Programming Tutor.
1983.
Journal of Computer-Based Instruction, in press.

What do Novices Know about Programming.
In A. Badre and B. Shneiderman (editor), Directions in Human-Computer Interaction.

[18] Soloway, E., Ehrlich, K., and Gold, E.
Reading a Program Is Like Reading a Story (Well, Almost).
Rochester, N.Y., 1983.

1983.
Communications of the ACM, in press.

Knowledge-based CAI: CINS for Individualized Curriculum Sequencing.
Psychology and Education Series, October, 1977.
- OFFICIAL DISTRIBUTION LIST -

**Army**

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical Director</td>
<td>U.S. Army Research Institute</td>
<td>1</td>
</tr>
<tr>
<td>Mr. James Baker</td>
<td>5001 Eisenhower Avenue</td>
<td>1</td>
</tr>
<tr>
<td>Dr. Beatrice J. Farr</td>
<td>U.S. Army Research Institute</td>
<td>1</td>
</tr>
<tr>
<td>Dr. Milton S. Katz</td>
<td>Williams Technical Area</td>
<td>1</td>
</tr>
<tr>
<td>Dr. Marshall H. N.</td>
<td>U.S. Army Research Institute</td>
<td>1</td>
</tr>
<tr>
<td>Dr. Harold P. O'Neil, Jr.</td>
<td>Army Research Institute</td>
<td>1</td>
</tr>
<tr>
<td>Dr. Barbara Hayes-Roth</td>
<td>US Army Research Institute</td>
<td>1</td>
</tr>
<tr>
<td>Joseph Paola, Ph.D.</td>
<td>PERI-BR (Dr. Judith Orasanu)</td>
<td>1</td>
</tr>
<tr>
<td>Dr. Robert Samor</td>
<td>Army Research Institute</td>
<td>1</td>
</tr>
<tr>
<td>Dr. Robert Wisinger</td>
<td>Army Research Institute</td>
<td>1</td>
</tr>
</tbody>
</table>

**Private Sector**

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dr. Michael Geesereth</td>
<td>Department of Computer Science</td>
<td>1</td>
</tr>
<tr>
<td>Dr. Fred Hayes-Roth</td>
<td>Teknowledge</td>
<td>1</td>
</tr>
<tr>
<td>Dr. James G. Greeno</td>
<td>LROC</td>
<td>1</td>
</tr>
<tr>
<td>Dr. Joseph Oates</td>
<td>SRI International</td>
<td>1</td>
</tr>
<tr>
<td>Dr. Bert Green</td>
<td>Johns Hopkins University</td>
<td>1</td>
</tr>
<tr>
<td>Dr. Harold F. O'Neil, Jr.</td>
<td>Army Research Institute</td>
<td>1</td>
</tr>
<tr>
<td>Dr. Barbara Hayes-Roth</td>
<td>Department of Psychology</td>
<td>1</td>
</tr>
<tr>
<td>Dr. Frederick Hayes-Roth</td>
<td>University of Pittsburgh</td>
<td>1</td>
</tr>
<tr>
<td>Dr. Earl Hunt</td>
<td>Department of Psychology</td>
<td>1</td>
</tr>
<tr>
<td>Dr. Marcel Just</td>
<td>University of Washington</td>
<td>1</td>
</tr>
</tbody>
</table>

**Locations**

- Stanford University, California 94305
- Stanford, California 94305
- University of Pittsburgh, Pennsylvania 15260
- University of Pittsburgh, Pennsylvania 15213
- University of Stanford, California 94025
- Johns Hopkins University, Maryland 21218
- University of Pittsburgh, Pennsylvania 15213
- Pittsburgh, Pennsylvania 15213
- University of Pittsburgh, California 94301
- University of Pittsburgh, Michigan 48012
- University of Pittsburgh, Washington 98105
- University of Pittsburgh, Pennsylvania 15213
- Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213
Air Force

U.S. Air Force Office of Scientific Research
Life Sciences Directorate, AFSC
Boeing Air Force Base
Washington, DC 20332

Dr. Earl A. Allee
HQ AFRL, (AFSC)
Brooks AFB, Texas 78235

Dr. David Kieras
Department of Psychology
University of Arizona
Tucson, Arizona 85721

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

Bryan Dallman
AFRL/LRT
Lowry AFB, Colorado 80230

Dr. Stephen Kosslyn
Department of Psychology
The Johns Hopkins University
Baltimore, Maryland 21218

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

Dr. Pat Langley
The Robotics Institute
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Dr. John Tangney
AFOSR/NL
Bolinger AFB, DC 20332

Dr. Jill Larkin
Department of Psychology
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Dr. Joseph Yasutake
AFRL/LRT
Lowry AFB, Colorado 80230

Dr. Alan Lengold
Learning R&D Center
University of Pittsburgh
3559 O'Hara Street
Pittsburgh, Pennsylvania 15213

Dr. William Greeneup
Education Advisor (EO31)
Education Center, MCDEC
Quantico, Virginia 22134

Dr. Jim Lewis
University of California
at San Diego
Laboratory for Comparative Human Cognition – D003A
La Jolla, California 92038

Dr. Michael Levine
Department of Educational Psychology
210 Education Bldg
University of Illinois
Champaign, Illinois 61801

Dr. Dana L. Saffoom
Scientific Advisor (Code RD-1)
HQ, U.S. Marine Corps
Washington, DC 20380

Dr. Marcia Lie
University of California
Director, Adolescent Reasoning Project
Berkeley, California 94720

Department of Defense

Defense Technical Information Center
Cameron Station, Bldg 5
Alexandria, Virginia 22314

Dr. Jay McClelland
Department of Psychology
MIT
Cambridge, Massachusetts 02139

Dr. Mark Miller
Computer Thought Corporation
1721 West Plano Highway
Plano, Texas 75075

Dr. James R. Miller
Computer Thought Corporation
1721 West Plano Highway
Plano, Texas 75075

Department of Defense

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

Dr. Mark Miller
Computer Thought Corporation
1721 West Plano Highway
Plano, Texas 75075

Major Jack Thorpe
DARPA
1400 Wilson Blvd
Arlington, Virginia 22209

Dr. William McRobie
Naval Research Laboratory
Washington, DC 20374

Dr. Michael Levine
Department of Psychology
210 Education Bldg
University of Illinois
Champaign, Illinois 61801

Dr. Samuel H. Wickens
Naval Research Laboratory
Washington, DC 20374

Dr. Mark Miller
Computer Thought Corporation
1721 West Plano Highway
Plano, Texas 75075

Dr. David Kieras
Department of Psychology
University of Arizona
Tucson, Arizona 85721

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

Bryan Dallman
AFRL/LRT
Lowry AFB, Colorado 80230

Dr. Stephen Kosslyn
Department of Psychology
The Johns Hopkins University
Baltimore, Maryland 21218

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

Dr. Pat Langley
The Robotics Institute
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Dr. John Tangney
AFOSR/NL
Bolinger AFB, DC 20332

Dr. Jill Larkin
Department of Psychology
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Dr. Joseph Yasutake
AFRL/LRT
Lowry AFB, Colorado 80230

Dr. Alan Lengold
Learning R&D Center
University of Pittsburgh
3559 O'Hara Street
Pittsburgh, Pennsylvania 15213

Dr. William Greeneup
Education Advisor (EO31)
Education Center, MCDEC
Quantico, Virginia 22134

Dr. Jim Lewis
University of California
at San Diego
Laboratory for Comparative Human Cognition – D003A
La Jolla, California 92038

Dr. Michael Levine
Department of Educational Psychology
210 Education Bldg
University of Illinois
Champaign, Illinois 61801

Dr. Dana L. Saffoom
Scientific Advisor (Code RD-1)
HQ, U.S. Marine Corps
Washington, DC 20380

Dr. Marcia Lie
University of California
Director, Adolescent Reasoning Project
Berkeley, California 94720

Dr. Jay McClelland
Department of Psychology
MIT
Cambridge, Massachusetts 02139

Major Jack Thorpe
DARPA
1400 Wilson Blvd
Arlington, Virginia 22209
Navy

Roberts Avenues
Code 8711
Human Factors Laboratory
Raytheon
Orlando, Florida 32813

Code 8711
Attn: Arthur S. Bliesner
Naval Training Equipment Center
Orlando, Florida 32813

Liaison Scientist
Office of Naval Research
Branch Office, London

Box 38
FPD New York, New York 08510

Dr. Richard Castone
Navy Research Laboratory
Code 7510
Washington, DC 20375

Chief of Naval Education and Training
Liaison Office
Air Force Human Resource Laboratory
Operations Training Division
WILLIAMS AFB, Arizona 85224

Dr. Stanley Collyer
Office of Naval Technology
800 N. Quincy Street
Arlington, Virginia 22217

CDR Mike Corras
Office of Naval Research
800 N. Quincy Street
Code 279
Arlington, Virginia 22217

Dr. John Ford
Navy Personnel R&D Center
San Diego, California 92132

Dr. Jade Franklin
Code 7510
Navy Research Laboratory
Washington, DC 20375

Dr. Mike Geyser
Navy Research Laboratory
Code 7510
Washington, DC 20375

Dr. Jim Holman
Code 14
Navy Personnel R&D Center
San Diego, California 92132

Dr. Ed Saleah
Navy Personnel R&D Center
San Diego, California 92132

Dr. Tom Morus
Xerox PARC
3333 Coyote Hill Road
Palo Alto, California 94304

Dr. Allen Haaro
Behavioral Technology Laboratories
1945 Eton Avenue, Fourth Floor
Redwood Beach, California 90277

Dr. Donald Norman
Cognitive Science, C-015
Univ of California, San Diego
La Jolla, California 92033

Dr. Jesse Orlandy
Institute for Defense Analyses
1991 N. Beauregard Street
Alexandria, Virginia 22311

Professor Seymour Papert
MIT
Cambridge, Massachusetts 02139

Dr. Nancy Pennington
University of Chicago
Graduate School of Business
1101 E 58th Street
Chicago, Illinois 60637

Dr. Richard A. Pollak
Director, Special Projects
MECC
2354 Hidden Valley Lane
Stillwater, Minnesota 55082

Dr. Fred Reif
Department of Psychology
University of Colorado
Boulder, Colorado 80309

Dr. Lauren Resnick
LRDC
University of Pittsburgh
3330 Cool Street
Pittsburgh, Pennsylvania 15213

Mary S. Riley
Program in Cognitive Science
Center for Human Information Processing
University of California, San Diego
La Jolla, California 92033

Dr. Andrew Rone
American Institutes for Research
1055 Thomas Jefferson Street, NW
Washington, DC 20007
<table>
<thead>
<tr>
<th>Name</th>
<th>Office/Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dr. Gil Ricard</td>
<td>Code N711</td>
</tr>
<tr>
<td>NTEC</td>
<td>Orlando, Florida 32813</td>
</tr>
<tr>
<td>Dr. Worth Scotland</td>
<td>CNET (N-5)</td>
</tr>
<tr>
<td>NAS, Pensacola, Florida 32508</td>
<td></td>
</tr>
<tr>
<td>Dr. Robert C. Smith</td>
<td>Office of Chief of Naval Operations</td>
</tr>
<tr>
<td>DP-607M</td>
<td>Washington, DC 20350</td>
</tr>
<tr>
<td>Dr. Alfred F. Smole, Director</td>
<td>Training Analysis &amp; Evaluation Group</td>
</tr>
<tr>
<td>Mr. John Wolfe</td>
<td>Navy Personnel R&amp;D Center</td>
</tr>
<tr>
<td>San Diego, California 92152</td>
<td></td>
</tr>
<tr>
<td>Dr. Frederick Steinheiser</td>
<td>CMD - OP115</td>
</tr>
<tr>
<td>Navy Annex</td>
<td>Arlington, Virginia 20270</td>
</tr>
<tr>
<td>Dr. Richard Sorenes</td>
<td>Department of Administrative Sciences</td>
</tr>
<tr>
<td>Naval Postgraduate School</td>
<td>Monterey, California 93940</td>
</tr>
<tr>
<td>Dr. Maurice Tatsnoka</td>
<td>Computer Based Education Research Laboratory</td>
</tr>
<tr>
<td>Dr. Keith T. Wescourt</td>
<td>University of Warwick</td>
</tr>
<tr>
<td>Dr. John R. Anderson</td>
<td>Department of Psychology</td>
</tr>
<tr>
<td>Carnegie-Mellon University</td>
<td>Pittsburgh, Pennsylvania 15213</td>
</tr>
<tr>
<td>Dr. John Asnett</td>
<td>Department of Psychology</td>
</tr>
<tr>
<td>University of Warwick</td>
<td>Coventry CV4 7AJ</td>
</tr>
<tr>
<td>Dr. Keith T. Wescourt</td>
<td>Applied Psychology Unit</td>
</tr>
<tr>
<td>Dr. Michael Atwood</td>
<td>ITT - Programming</td>
</tr>
<tr>
<td>1066 Groveque Lane</td>
<td>Stratford, Connecticut 06497</td>
</tr>
<tr>
<td>Dr. Alan Baddeley</td>
<td>Medical Research Council</td>
</tr>
<tr>
<td>Applied Psychology Unit</td>
<td>15 Chester Road</td>
</tr>
<tr>
<td>Cambridge CB2 2EF</td>
<td></td>
</tr>
<tr>
<td>Dr. Robert Sternberg</td>
<td>Department of Psychology</td>
</tr>
<tr>
<td>Yale University</td>
<td>Box 11A, Yale Station</td>
</tr>
<tr>
<td>New Haven, Connecticut 06520</td>
<td></td>
</tr>
<tr>
<td>Dr. Albert Stevens</td>
<td>Bolt Beranek &amp; Newman</td>
</tr>
<tr>
<td>10 Monitor Street</td>
<td>Cambridge, Massachusetts 02238</td>
</tr>
<tr>
<td>Dr. Robert C. Smith</td>
<td>Department of Psychology</td>
</tr>
<tr>
<td>Yale University</td>
<td>Box 11A, Yale Station</td>
</tr>
<tr>
<td>New Haven, Connecticut 06520</td>
<td></td>
</tr>
<tr>
<td>Dr. Patrick Suppes</td>
<td>Institute for Mathematical Studies in the Social</td>
</tr>
<tr>
<td>Stanford University</td>
<td>Stanford, California 94305</td>
</tr>
<tr>
<td>Dr. Maurice Tatsnoka</td>
<td>220 Education Bldg</td>
</tr>
<tr>
<td>Dr. Keith T. Wescourt</td>
<td>545 Middlefield Road, Suite 140</td>
</tr>
<tr>
<td>Dr. Douglas Towne</td>
<td>University of Southern California Behavioral</td>
</tr>
<tr>
<td>Dr. John R. Anderson</td>
<td>Applied Psychology Unit</td>
</tr>
<tr>
<td>Carnegie-Mellon University</td>
<td>Pittsburgh, Pennsylvania 15213</td>
</tr>
<tr>
<td>Dr. Michael Atwood</td>
<td>ITT - Programming</td>
</tr>
<tr>
<td>1066 Groveque Lane</td>
<td>Stratford, Connecticut 06497</td>
</tr>
<tr>
<td>Dr. Alan Baddeley</td>
<td>Medical Research Council</td>
</tr>
<tr>
<td>Applied Psychology Unit</td>
<td>15 Chester Road</td>
</tr>
<tr>
<td>Cambridge CB2 2EF</td>
<td></td>
</tr>
<tr>
<td>Dr. Robert C. Smith</td>
<td>Office of Chief of Naval Operations</td>
</tr>
<tr>
<td>DP-607M</td>
<td>Washington, DC 20350</td>
</tr>
<tr>
<td>Dr. Alfred F. Smole, Director</td>
<td>Training Analysis &amp; Evaluation Group</td>
</tr>
<tr>
<td>Mr. John Wolfe</td>
<td>Navy Personnel R&amp;D Center</td>
</tr>
<tr>
<td>San Diego, California 92152</td>
<td></td>
</tr>
<tr>
<td>Dr. Frederick Steinheiser</td>
<td>CMD - OP115</td>
</tr>
<tr>
<td>Navy Annex</td>
<td>Arlington, Virginia 20270</td>
</tr>
<tr>
<td>Dr. Richard Sorenes</td>
<td>Department of Administrative Sciences</td>
</tr>
<tr>
<td>Naval Postgraduate School</td>
<td>Monterey, California 93940</td>
</tr>
<tr>
<td>Dr. Maurice Tatsnoka</td>
<td>Computer Based Education Research Laboratory</td>
</tr>
<tr>
<td>Dr. Keith T. Wescourt</td>
<td>University of Southern California Behavioral</td>
</tr>
<tr>
<td>Dr. Douglas Towne</td>
<td>Applied Psychology Unit</td>
</tr>
<tr>
<td>Dr. John R. Anderson</td>
<td>Department of Psychology</td>
</tr>
<tr>
<td>Carnegie-Mellon University</td>
<td>Pittsburgh, Pennsylvania 15213</td>
</tr>
<tr>
<td>Dr. Michael Atwood</td>
<td>ITT - Programming</td>
</tr>
<tr>
<td>1066 Groveque Lane</td>
<td>Stratford, Connecticut 06497</td>
</tr>
<tr>
<td>Dr. Alan Baddeley</td>
<td>Medical Research Council</td>
</tr>
<tr>
<td>Applied Psychology Unit</td>
<td>15 Chester Road</td>
</tr>
<tr>
<td>Cambridge CB2 2EF</td>
<td></td>
</tr>
<tr>
<td>Dr. Robert C. Smith</td>
<td>Department of Psychology</td>
</tr>
<tr>
<td>Yale University</td>
<td>Box 11A, Yale Station</td>
</tr>
<tr>
<td>New Haven, Connecticut 06520</td>
<td></td>
</tr>
<tr>
<td>Dr. Albert Stevens</td>
<td>Bolt Beranek &amp; Newman</td>
</tr>
<tr>
<td>10 Monitor Street</td>
<td>Cambridge, Massachusetts 02238</td>
</tr>
<tr>
<td>Dr. Patrick Suppes</td>
<td>Institute for Mathematical Studies in the Social</td>
</tr>
<tr>
<td>Stanford University</td>
<td>Stanford, California 94305</td>
</tr>
<tr>
<td>Dr. Maurice Tatsnoka</td>
<td>220 Education Bldg</td>
</tr>
<tr>
<td>Dr. Keith T. Wescourt</td>
<td>545 Middlefield Road, Suite 140</td>
</tr>
<tr>
<td>Dr. Douglas Towne</td>
<td>University of Southern California Behavioral</td>
</tr>
<tr>
<td>Dr. John R. Anderson</td>
<td>Department of Psychology</td>
</tr>
<tr>
<td>Carnegie-Mellon University</td>
<td>Pittsburgh, Pennsylvania 15213</td>
</tr>
<tr>
<td>Dr. Michael Atwood</td>
<td>ITT - Programming</td>
</tr>
<tr>
<td>1066 Groveque Lane</td>
<td>Stratford, Connecticut 06497</td>
</tr>
<tr>
<td>Dr. Alan Baddeley</td>
<td>Medical Research Council</td>
</tr>
<tr>
<td>Applied Psychology Unit</td>
<td>15 Chester Road</td>
</tr>
<tr>
<td>Cambridge CB2 2EF</td>
<td></td>
</tr>
<tr>
<td>Dr. Robert C. Smith</td>
<td>Office of Chief of Naval Operations</td>
</tr>
<tr>
<td>DP-607M</td>
<td>Washington, DC 20350</td>
</tr>
<tr>
<td>Dr. Alfred F. Smole, Director</td>
<td>Training Analysis &amp; Evaluation Group</td>
</tr>
<tr>
<td>Mr. John Wolfe</td>
<td>Navy Personnel R&amp;D Center</td>
</tr>
<tr>
<td>San Diego, California 92152</td>
<td></td>
</tr>
<tr>
<td>Dr. Frederick Steinheiser</td>
<td>CMD - OP115</td>
</tr>
<tr>
<td>Navy Annex</td>
<td>Arlington, Virginia 20270</td>
</tr>
<tr>
<td>Dr. Richard Sorenes</td>
<td>Department of Administrative Sciences</td>
</tr>
<tr>
<td>Naval Postgraduate School</td>
<td>Monterey, California 93940</td>
</tr>
<tr>
<td>Dr. Maurice Tatsnoka</td>
<td>Computer Based Education Research Laboratory</td>
</tr>
<tr>
<td>Dr. Keith T. Wescourt</td>
<td>University of Southern California Behavioral</td>
</tr>
<tr>
<td>Dr. Douglas Towne</td>
<td>Applied Psychology Unit</td>
</tr>
<tr>
<td>Dr. John R. Anderson</td>
<td>Department of Psychology</td>
</tr>
<tr>
<td>Carnegie-Mellon University</td>
<td>Pittsburgh, Pennsylvania 15213</td>
</tr>
<tr>
<td>Dr. Michael Atwood</td>
<td>ITT - Programming</td>
</tr>
<tr>
<td>1066 Groveque Lane</td>
<td>Stratford, Connecticut 06497</td>
</tr>
<tr>
<td>Dr. Alan Baddeley</td>
<td>Medical Research Council</td>
</tr>
<tr>
<td>Applied Psychology Unit</td>
<td>15 Chester Road</td>
</tr>
<tr>
<td>Cambridge CB2 2EF</td>
<td></td>
</tr>
<tr>
<td>Dr. Robert C. Smith</td>
<td>Office of Chief of Naval Operations</td>
</tr>
<tr>
<td>DP-607M</td>
<td>Washington, DC 20350</td>
</tr>
<tr>
<td>Dr. Alfred F. Smole, Director</td>
<td>Training Analysis &amp; Evaluation Group</td>
</tr>
<tr>
<td>Mr. John Wolfe</td>
<td>Navy Personnel R&amp;D Center</td>
</tr>
<tr>
<td>San Diego, California 92152</td>
<td></td>
</tr>
<tr>
<td>Dr. Frederick Steinheiser</td>
<td>CMD - OP115</td>
</tr>
<tr>
<td>Navy Annex</td>
<td>Arlington, Virginia 20270</td>
</tr>
<tr>
<td>Dr. Richard Sorenes</td>
<td>Department of Administrative Sciences</td>
</tr>
<tr>
<td>Naval Postgraduate School</td>
<td>Monterey, California 93940</td>
</tr>
<tr>
<td>Dr. Maurice Tatsnoka</td>
<td>Computer Based Education Research Laboratory</td>
</tr>
<tr>
<td>Dr. Keith T. Wescourt</td>
<td>University of Southern California Behavioral</td>
</tr>
<tr>
<td>Dr. Douglas Towne</td>
<td>Applied Psychology Unit</td>
</tr>
<tr>
<td>Dr. John R. Anderson</td>
<td>Department of Psychology</td>
</tr>
<tr>
<td>Carnegie-Mellon University</td>
<td>Pittsburgh, Pennsylvania 15213</td>
</tr>
<tr>
<td>Dr. Michael Atwood</td>
<td>ITT - Programming</td>
</tr>
<tr>
<td>1066 Groveque Lane</td>
<td>Stratford, Connecticut 06497</td>
</tr>
<tr>
<td>Dr. Alan Baddeley</td>
<td>Medical Research Council</td>
</tr>
<tr>
<td>Applied Psychology Unit</td>
<td>15 Chester Road</td>
</tr>
<tr>
<td>Cambridge CB2 2EF</td>
<td></td>
</tr>
<tr>
<td>Dr. Robert C. Smith</td>
<td>Office of Chief of Naval Operations</td>
</tr>
<tr>
<td>DP-607M</td>
<td>Washington, DC 20350</td>
</tr>
<tr>
<td>Dr. Alfred F. Smole, Director</td>
<td>Training Analysis &amp; Evaluation Group</td>
</tr>
<tr>
<td>Mr. John Wolfe</td>
<td>Navy Personnel R&amp;D Center</td>
</tr>
<tr>
<td>San Diego, California 92152</td>
<td></td>
</tr>
<tr>
<td>Dr. Frederick Steinheiser</td>
<td>CMD - OP115</td>
</tr>
<tr>
<td>Navy Annex</td>
<td>Arlington, Virginia 20270</td>
</tr>
<tr>
<td>Dr. Richard Sorenes</td>
<td>Department of Administrative Sciences</td>
</tr>
<tr>
<td>Naval Postgraduate School</td>
<td>Monterey, California 93940</td>
</tr>
<tr>
<td>Dr. Maurice Tatsnoka</td>
<td>Computer Based Education Research Laboratory</td>
</tr>
<tr>
<td>Dr. Keith T. Wescourt</td>
<td>University of Southern California Behavioral</td>
</tr>
<tr>
<td>Dr. Douglas Towne</td>
<td>Applied Psychology Unit</td>
</tr>
<tr>
<td>Dr. John R. Anderson</td>
<td>Department of Psychology</td>
</tr>
<tr>
<td>Carnegie-Mellon University</td>
<td>Pittsburgh, Pennsylvania 15213</td>
</tr>
<tr>
<td>Dr. Michael Atwood</td>
<td>ITT - Programming</td>
</tr>
<tr>
<td>1066 Groveque Lane</td>
<td>Stratford, Connecticut 06497</td>
</tr>
<tr>
<td>Dr. Alan Baddeley</td>
<td>Medical Research Council</td>
</tr>
<tr>
<td>Applied Psychology Unit</td>
<td>15 Chester Road</td>
</tr>
<tr>
<td>Cambridge CB2 2EF</td>
<td></td>
</tr>
<tr>
<td>Dr. Robert C. Smith</td>
<td>Office of Chief of Naval Operations</td>
</tr>
<tr>
<td>DP-607M</td>
<td>Washington, DC 20350</td>
</tr>
<tr>
<td>Dr. Alfred F. Smole, Director</td>
<td>Training Analysis &amp; Evaluation Group</td>
</tr>
<tr>
<td>Mr. John Wolfe</td>
<td>Navy Personnel R&amp;D Center</td>
</tr>
<tr>
<td>San Diego, California 92152</td>
<td></td>
</tr>
<tr>
<td>Dr. Frederick Steinheiser</td>
<td>CMD - OP115</td>
</tr>
<tr>
<td>Navy Annex</td>
<td>Arlington, Virginia 20270</td>
</tr>
<tr>
<td>Dr. Richard Sorenes</td>
<td>Department of Administrative Sciences</td>
</tr>
<tr>
<td>Naval Postgraduate School</td>
<td>Monterey, California 93940</td>
</tr>
<tr>
<td>Dr. Maurice Tatsnoka</td>
<td>Computer Based Education Research Laboratory</td>
</tr>
<tr>
<td>Dr. Keith T. Wescourt</td>
<td>University of Southern California Behavioral</td>
</tr>
<tr>
<td>Dr. Douglas Towne</td>
<td>Applied Psychology Unit</td>
</tr>
<tr>
<td>Dr. John R. Anderson</td>
<td>Department of Psychology</td>
</tr>
<tr>
<td>Carnegie-Mellon University</td>
<td>Pittsburgh, Pennsylvania 15213</td>
</tr>
<tr>
<td>Dr. Michael Atwood</td>
<td>ITT - Programming</td>
</tr>
<tr>
<td>1066 Groveque Lane</td>
<td>Stratford, Connecticut 06497</td>
</tr>
<tr>
<td>Dr. Alan Baddeley</td>
<td>Medical Research Council</td>
</tr>
<tr>
<td>Applied Psychology Unit</td>
<td>15 Chester Road</td>
</tr>
<tr>
<td>Cambridge CB2 2EF</td>
<td></td>
</tr>
<tr>
<td>Civilian Agencies</td>
<td>1 copy</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>Dr Patricia Baggett</td>
<td>1 copy</td>
</tr>
<tr>
<td>Department of Psychology</td>
<td></td>
</tr>
<tr>
<td>University of Colorado</td>
<td></td>
</tr>
<tr>
<td>Boulder, Colorado 80309</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Ms Carol A Bagley</td>
<td>1 copy</td>
</tr>
<tr>
<td>Minnesota Educational Computing Consortium</td>
<td></td>
</tr>
<tr>
<td>2354 Hidden Valley Lane</td>
<td></td>
</tr>
<tr>
<td>Stillwater, Minnesota 55082</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Dr Jonathan Barr</td>
<td>1 copy</td>
</tr>
<tr>
<td>80 Glenn Avenue</td>
<td></td>
</tr>
<tr>
<td>Berwyn, Pennsylvania 19312</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Mr Avron Barr</td>
<td>1 copy</td>
</tr>
<tr>
<td>Department of Computer Science</td>
<td></td>
</tr>
<tr>
<td>Stanford University</td>
<td></td>
</tr>
<tr>
<td>Stanford, California 94305</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Dr John Black</td>
<td>1 copy</td>
</tr>
<tr>
<td>Yale University</td>
<td></td>
</tr>
<tr>
<td>Box 11A Yale Station</td>
<td></td>
</tr>
<tr>
<td>New Haven, Connecticut 06520</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Dr John S Brown</td>
<td>1 copy</td>
</tr>
<tr>
<td>Xerox Palo Alto Research Center</td>
<td></td>
</tr>
<tr>
<td>3333 Coyote Road</td>
<td></td>
</tr>
<tr>
<td>Palo Alto, California 94304</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Dr Bruce Buchanan</td>
<td>1 copy</td>
</tr>
<tr>
<td>Department of Computer Science</td>
<td></td>
</tr>
<tr>
<td>Stanford University</td>
<td></td>
</tr>
<tr>
<td>Stanford, California 94305</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Dr Jaime Carbonell</td>
<td>1 copy</td>
</tr>
<tr>
<td>Department of Psychology</td>
<td></td>
</tr>
<tr>
<td>Carnegie-Mellon University</td>
<td></td>
</tr>
<tr>
<td>Pittsburgh, Pennsylvania 15213</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Dr Pat Carpenter</td>
<td>1 copy</td>
</tr>
<tr>
<td>Department of Psychology</td>
<td></td>
</tr>
<tr>
<td>Carnegie-Mellon University</td>
<td></td>
</tr>
<tr>
<td>Pittsburgh, Pennsylvania 15213</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Dr William Chase</td>
<td>1 copy</td>
</tr>
<tr>
<td>Department of Psychology</td>
<td></td>
</tr>
<tr>
<td>Carnegie-Mellon University</td>
<td></td>
</tr>
<tr>
<td>Pittsburgh, Pennsylvania 15213</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Dr Michelle Chi</td>
<td>1 copy</td>
</tr>
<tr>
<td>Learning R &amp; D Center</td>
<td></td>
</tr>
<tr>
<td>University of Pittsburgh</td>
<td></td>
</tr>
<tr>
<td>3355 O'Hara Street</td>
<td></td>
</tr>
<tr>
<td>Pittsburgh, Pennsylvania 15213</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Dr Patricia A Belter</td>
<td>1 copy</td>
</tr>
<tr>
<td>NIE-BRN Bldg, Stop 87</td>
<td></td>
</tr>
<tr>
<td>1200 19th Street NW</td>
<td></td>
</tr>
<tr>
<td>Washington, DC 20208</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Dr Susan Chipman</td>
<td>1 copy</td>
</tr>
<tr>
<td>Learning and Development</td>
<td></td>
</tr>
<tr>
<td>National Institute of Education</td>
<td></td>
</tr>
<tr>
<td>1200 19th Street NW</td>
<td></td>
</tr>
<tr>
<td>Washington, DC 20208</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Edward Eady</td>
<td>1 copy</td>
</tr>
<tr>
<td>Department of Education, OERI</td>
<td></td>
</tr>
<tr>
<td>MS 40</td>
<td></td>
</tr>
<tr>
<td>1200 19th Street NW</td>
<td></td>
</tr>
<tr>
<td>Washington, DC 20208</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Edward J Fuentes</td>
<td>1 copy</td>
</tr>
<tr>
<td>Department of Education</td>
<td></td>
</tr>
<tr>
<td>1200 19th Street NW</td>
<td></td>
</tr>
<tr>
<td>Washington, DC 20208</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TAEX TIX</td>
<td>1 copy</td>
</tr>
<tr>
<td>National Institute of Education</td>
<td></td>
</tr>
<tr>
<td>1200 19th Street NW</td>
<td></td>
</tr>
<tr>
<td>Washington, DC 20208</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Dr John Hayes</td>
<td>1 copy</td>
</tr>
<tr>
<td>National Institute of Education</td>
<td></td>
</tr>
<tr>
<td>1200 19th Street NW</td>
<td></td>
</tr>
<tr>
<td>Washington, DC 20208</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Dr Arthur Melmed</td>
<td>1 copy</td>
</tr>
<tr>
<td>U S Dept of Education</td>
<td></td>
</tr>
<tr>
<td>Washington, DC 20208</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Dr Andrew R Molnar</td>
<td>1 copy</td>
</tr>
<tr>
<td>Office of Scientific and Engineering Personnel and Education</td>
<td></td>
</tr>
<tr>
<td>National Science Foundation</td>
<td></td>
</tr>
<tr>
<td>Washington, DC 20550</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Everett Palmer</td>
<td>1 copy</td>
</tr>
<tr>
<td>Research Scientist</td>
<td></td>
</tr>
<tr>
<td>Mail Slope 399-3</td>
<td></td>
</tr>
<tr>
<td>NASA Ames Research Center</td>
<td></td>
</tr>
<tr>
<td>Moffett Field, California 94035</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Dr Mary Stoddard</td>
<td>1 copy</td>
</tr>
<tr>
<td>C 10, Mail Stop B296</td>
<td></td>
</tr>
<tr>
<td>Los Alamos National Laboratories</td>
<td></td>
</tr>
<tr>
<td>Los Alamos, New Mexico 87545</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Chief, Psychological Research Branch</td>
<td>1 copy</td>
</tr>
<tr>
<td>U s Coast Guard (OP-1/2/TNP2)</td>
<td></td>
</tr>
<tr>
<td>Washington, DC 20593</td>
<td></td>
</tr>
</tbody>
</table>