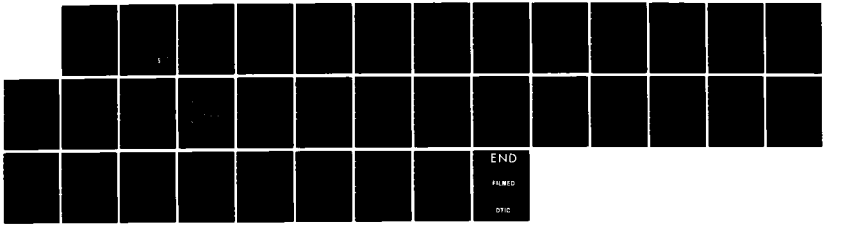
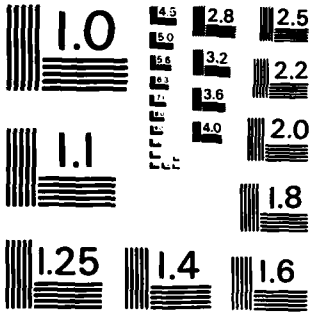
AD-A147 694 APPROACHES TO STRUCTURING THE SOFTWARE DEVELOPMENT PROCESS(U) GENERAL ELECTRIC CO ARLINGTON VA DATA AND INFORMATION SYSTEMS.. D A BOEHM-DAVIS ET AL. OCT 84 1/1

UNCLASSIFIED GEC/DIS/TR-84-81V-1 N00014-83-C-0574 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

GEC/DIS/TR-84-B1V-1

APPROACHES TO STRUCTURING THE SOFTWARE DEVELOPMENT PROCESS

Deborah A. Boehm-Davis
Lyle S. Ross

Software Management Research
Data & Information Systems
General Electric Company
1755 Jefferson Davis Highway
Arlington, Virginia 22202

October 1984

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: TR-84-B1V-1	2. GOVT ACCESSION NO.: AD-A147694
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): Approaches to Structuring the Software Development Process	5. TYPE OF REPORT & PERIOD COVERED: Technical Report (15 JUL 83 to 30 SEP 84)
	6. PERFORMING ORG. REPORT NUMBER: GEC/DIS/TR-84-B1V-1
7. AUTHOR(s): Deborah A. Boehm-Davis, Lyle S. Ross	8. CONTRACT OR GRANT NUMBER(s): N00014-83-C-0574
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Data & Information Systems, General Electric Company, 1755 Jefferson Davis Hwy., Arlington, VA 22202	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61153N 42; RR04209; RR04209; NR 196-183
11. CONTROLLING OFFICE NAME AND ADDRESS: Engineering Psychology Program, Code 442, Office of Naval Research, Arlington, VA 22217	12. REPORT DATE: October 1984
	13. NUMBER OF PAGES: 29
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): Same	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)

Same

18. SUPPLEMENTARY NOTES

Technical monitor: Dr. John J. O'Hare

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

Software engineering, Software experiments, Modern programming practices, Program design methodologies, Software human factors, Functional decomposition, Jackson Program Design Methodology, Object-Oriented Design

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

This research examined program design methodologies, which claim to improve the design process by providing strategies to programmers for structuring solutions to computer problems. In this experiment, professional programmers were provided with the specifications for each of three non-trivial problems and asked to produce pseudo-code for each specification according to the principles of a particular design methodology. The measures collected were the time to design and code, percent complete, and complexity, as measured

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)

by several metrics. These data were used to develop profiles of the solutions produced by different methodologies and to develop comparisons between the methodologies. The results suggest that there are differences among the various methodologies. These differences are discussed in light of their impact on the comprehensibility, reliability, and maintainability of the programs produced.

GEC/DIS/TR-84-B1V-1

APPROACHES TO STRUCTURING THE SOFTWARE DEVELOPMENT PROCESS

DEBORAH A. BOEHM-DAVIS
LYLE S. ROSS

Software Management Research
Data & Information Systems
General Electric Company
1755 Jefferson Davis Highway
Arlington, Virginia 22202

Submitted to:

Office of Naval Research
Engineering Psychology Program
Arlington, Virginia

Contract: N00014-83-C-0574
Work Unit: NR 196-183

OCTOBER 1984

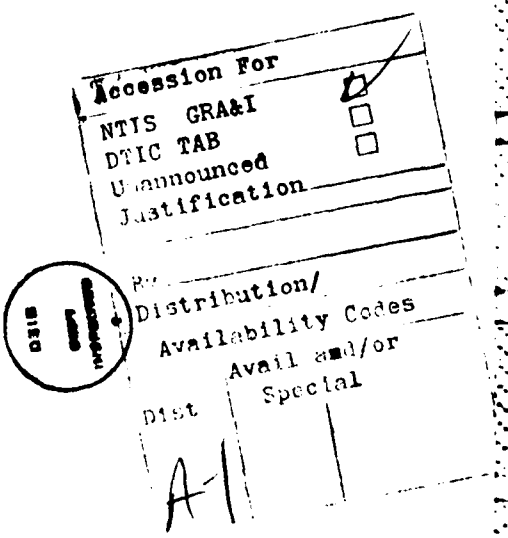


TABLE OF CONTENTS

INTRODUCTION

We have entered an era in which it has become increasingly important to develop human engineering principles which will significantly improve the structure of programs and assist programmers in ensuring system reliability. This is true, at least in part, because improved programs reduce labor costs, especially during later phases of the software life cycle where such costs are greatest (Putnam, 1978). Recent reports have asserted that almost 70% of costs associated with software are sustained after the product is delivered (Boehm, 1981). These costs generally are spent in maintenance; that is, modifications and error corrections to the original program. These figures suggest that even small improvements in program maintainability could be translated into substantial cost savings. While many methodologies, tools, and other programming aids have been developed to produce more maintainable software, little empirical work has been done to establish either objective measures of maintainability or a particular tool's success in producing a maintainable product.

Our recent series of studies investigating the impact of documentation format on program comprehensibility, codability, verifiability, and modifiability represents a systematic, objective evaluation of the impact of a programming tool (Boehm-Davis, Sheppard, Bailey & Kruesi, 1981; Sheppard, Bailey & Kruesi, 1981a, b; Sheppard & Kruesi, 1981; and Sheppard, Kruesi & Curtis, 1980). There is, however, almost a total absence of research examining the impact of tools and methodologies earlier in the process, such as in program design. Research done at TRW, IBM, and Raytheon suggests that errors made early in the project and carried on into testing and integration are the most costly type of error to find and correct. Also, characteristics of the program itself, such as its simplicity, generally determine the subsequent ease of understanding and modifying the program.

The psychological literature on problem solving has suggested that, at least for certain classes of problems, learning particular strategies for attempting solutions improves performance, especially for problem solvers of average ability. In the field of computer science, it has been argued that program design methodologies provide strategies to programmers for structuring solutions to computer problems. These methods seek to improve

the final program by dividing the problem into manageable parts, thus allowing the designer to deal with smaller units which are easier to code, verify, and modify. While some attempts have been made to compare specific design methodologies with each other, these comparisons have generally been non-experimental in nature and have not provided any general guidelines as to which methodologies (or which properties of methodologies) result in the most maintainable code. Such guidelines would be very useful for project managers. One approach to developing guidelines is to identify a major factor underlying the differences among methodologies and to evaluate the effect of this factor experimentally.

The basic difference among methodologies is the criterion used to decompose the problem into smaller units. The approaches basically vary along one dimension which is the extent to which the decomposition relies upon data structures as an organizing principle for modularization. On one end of the dimension are data structure techniques (such as Warnier's or Jackson's techniques) which rely primarily on the data structures present in the specifications as the basis for modularization. On the other end of the dimension are techniques which rely primarily on operations as the basis for structuring the problem (such as functional decomposition). In the former case, modules are organized around data structures while in the latter, modules are organized around operations. Falling between the two extremes are techniques which rely partially on data structures and partially on operations as the basis for structuring the programs (such as object-oriented design or Parnas' information hiding technique).

More specifically, in object-oriented design, the criterion used to modularize the program is the desire to create one module for each object (design decision) in the program. Operations are then defined for the object within that module and they are the only operations permitted on the object. In this way, each module can be independent from the other modules in the program (i.e., does not rely on knowledge of the representation of data in any other modules). In functional decomposition, the criterion used to modularize the program is the desire to create one module for each major processing step (or operation) in the program. High-level modules are then further decomposed into smaller modules, each of which represents a smaller processing step. Data structure techniques map input data to output data

and the criterion used to modularize the program is that each structure needed to map one input to one output becomes a module.

Using this dimension to classify methodologies, then, it was possible to generate programs decomposed in each of these ways, and to evaluate the effects of these decompositions in terms of the initial coding process and the quality of the resulting code. The focus of the research was on a comprehensive evaluation of programs produced by the different classes of methodologies. As such, the purpose of the evaluation was to determine the effects of methodology on variables such as time to design and code the programs, complexity of the solutions, and consistency among the solutions produced by each given methodology. These data allowed us to determine the relative cost to produce programs using given classes of methodologies as well as the quality of those programs, as measured by variables such as complexity.

METHOD

Design

The design of this experiment was a 3 X 3 design, where the independent variables were the decomposition technique and problems. The program decomposition technique was a between-subjects variable and problems were a within-subjects variable.

Materials

A set of specifications were developed for three programs. These were a naval air station problem, a host-at-sea buoy problem, and a military address problem. The problems were chosen such that they would require at least 100 to 200 lines of code, and therefore, require some decomposition in structuring their solution.

The naval air station problem involved determining the number of personnel using the facilities at a naval air station at any given time to ensure the most efficient use of the support facilities. The host-at-sea problem involved providing navigation and weather data to air and ship traffic at sea. In this problem, buoys are deployed to collect wind, temperature, and location data and they broadcast summaries of this information to passing vessels. The military address system involved maintaining a data base of names and postal addresses. From this data base, subsets of names and addressees could be drawn according to specified criteria and printed according to a specified format. The complete specifications for each of the problems can be seen in the Appendix.

Procedure

Programmers were provided with the specifications for each of three problems and asked to produce pseudo-code for each specification. Each time the programmers worked on the program, they were asked to complete a worksheet for the session. The worksheet asked (1) how much time the programmer had spent working on the problem since the last session, (2) how much time was spent during the current session, and (3) what was

accomplished during this session. In addition, the programmers were asked to append to the worksheet any rough drafts or scratch work generated during the session. These worksheets and scratch materials allowed us to track the software development process. At the end of the project, the participants were asked to complete a final questionnaire which provided us with information about each programmer's programming background, familiarity with the methodology, and reactions to the problems used in this research.

Participants

The participants in this research were 18 professional programmers. The programmers were volunteers from a number of different companies who were familiar with the requisite design methodology. It should be noted that some of the materials were incomplete when they were received. The analyses were performed on the available data, therefore, the number of participants is not equal in every analysis.

RESULTS

The measures collected in this research project fall into two general categories: objective measures and subjective evaluations of the solutions produced. The summary measures were completeness of the solution, time to design and code, experience, and complexity measures. The subjective evaluations focussed on the extent to which programs generated by the different methodologies differed, and the extent to which programs generated by a given methodology were alike.

Solution Completeness

The instructions provided to the programmers asked them to complete their designs in detailed pseudo-code. The intent was to have them provide pseudo-code at a low enough level that another programmer could write executable code from the pseudo-code without making any major design decisions. However, the level of detail in the pseudo-code received for analysis differed widely from programmer to programmer. As a result, the first analysis of the data involved calibrating the solutions according to a standard. This was done so that the data collected from different solutions could be compared on an equivalent scale.

For the analysis, the most detailed solution received was chosen as the standard and given the value of 100% complete. The level of detail in the other programs was compared to that standard and a percent complete value was assigned to each program relative to the standard. The average percent complete for the programs generated using each design methodology can be seen in Table 1. An analysis of variance confirmed that there were significant differences in percent complete given the design methodology used to generate the solution ($F(2,33) = 4.15$, $p < .05$).

Correlations performed between years of professional programming and completeness ($r = .36$) and between years of experience with the methodology and completeness ($r = .34$) were not significant.

DESIGN METHODOLOGY	PERCENT COMPLETE	McCABE METRIC	HALSTEAD METRIC (V)	LINES OF CODE	DESIGN TIME (HRS)	YEARS OF PROGRAMMING	YEARS WITH METHODOLOGY
JACKSON PROGRAM DESIGN	.82 (.15)	17 (4.3)	1142.5 (397.4)	114 (106.5)	12.2 (5.8)	15.0 (7.1)	4.7 (1.7)
OBJECT-ORIENTED DESIGN	.72 (.26)	32.7 (19.9)	9850.6 (18389.5	224.7 (209.5)	8.7 (6.2)	4.9 (3.6)	2.0 (1.4)
FUNCTIONAL DECOMPOSITION	.43 (.32)	18.9 (11.3)	2269.9 (1517.4)	90.8 (59.6)	11.5 (5.4)	3.4 (2.4)	2.3 (1.3)

TABLE 1. COMPARISON OF MEANS (AND STANDARD DEVIATIONS) OF DEPENDENT VARIABLES USING UNWEIGHTED RAW SCORES

Time to Design and Code

The time required to design and write the pseudo-code was computed from the worksheet data. These data are shown in Table 1, averaged across each program design methodology. An analysis of variance conducted on these data showed no significant differences in the time required to design and code the solutions ($F < 1$). In addition, the obtained data were weighted by the percent complete to obtain an estimate of the time that would have been required to generate a complete solution. These data are shown in Table 2; an analysis of variance confirmed a significant difference in the solution times ($F(2,31) = 3.58$, $p < .05$).

DESIGN METHODOLOGY	McCABE METRIC	HALSTEAD METRIC (V)	LINES OF CODE	DESIGN TIME (HRS)
JACKSON PROGRAM DESIGN	18.7 (15.1)	1320.9 (381.2)	127.2 (100.2)	14.7 (6.8)
OBJECT-ORIENTED DESIGN	43.2 (19.6)	13449.6 (22634.8)	301.2 (188.5)	13.1 (8.9)
FUNCTIONAL DECOMPOSITION	59.4 (30.2)	6177.7 (2535.7)	249.4 (92.8)	56.9 (50.3)

TABLE 2. COMPARISONS OF MEANS (AND STANDARD DEVIATIONS) OF DEPENDENT VARIABLES WEIGHTED BY PERCENT COMPLETE

Complexity Metrics

Three complexity metrics were calculated on each program completed: the McCabe metric (McCabe, 1976), the Halstead volume metric (see Curtis & Sheppard, 1979), and lines of code. The values of each metric calculated on the pseudo-code can be seen in Table 1. An analysis of variance showed no significant differences for the McCabe metric ($F(2,33) = 1.40$), the Halstead metric ($F(2,33) = 1.64$), or for the lines of code ($F < 1$) at the five percent level.

As with the design time measure, the complexity metrics were weighted by the percent complete, and these values are shown in Table 2. In this anlysis, significant differences were found for the McCabe metric ($F(2,33) = 4.84$) and for the lines of code ($F(2,33) = 3.58$), but not for the Halstead metric ($F(2,33) = 1.78$) at the five percent level.

Experience

Two measures of experience were collected from the participants, the number of years they had been programming professionally, and the number of years they had been working with the particular program design methodology they used. The mean values of these measures are shown in Table 1 for each design methodology. There was a significant difference in the overall years of experience ($F(2,10) = 7.99$) as well as in the number of years experience with the methodology ($F(2,10) = 5.13$).

Differences in Solutions

The definition of a program design methodology implies that the structure of the design solution produced by a given methodology should be different than one produced by another given methodology. This aspect of the solutions produced in this research was examined in a subjective fashion. The solutions received suggested that different design structures were being generated by the different methodologies. As an illustration, Figures 1, 2, and 3 provide sample structures for the solutions to the Military Address Problem generated by the functional decomposition, Jackson program design, and object-oriented approaches, respectively. It can be

seen that the approaches to the solution do differ. Specifically, the modules built around processing steps, or operations, in the functional decomposition solutions differ from the modules built around data structures in the Jackson program design methodology. Finally, the modules built around design decisions in the object-oriented approach differ from either of the first two approaches. This suggests that the solutions generated by using different program design methodologies do differ. It also suggests that the different methodologies are indeed emphasizing different decompositions of the problem space.

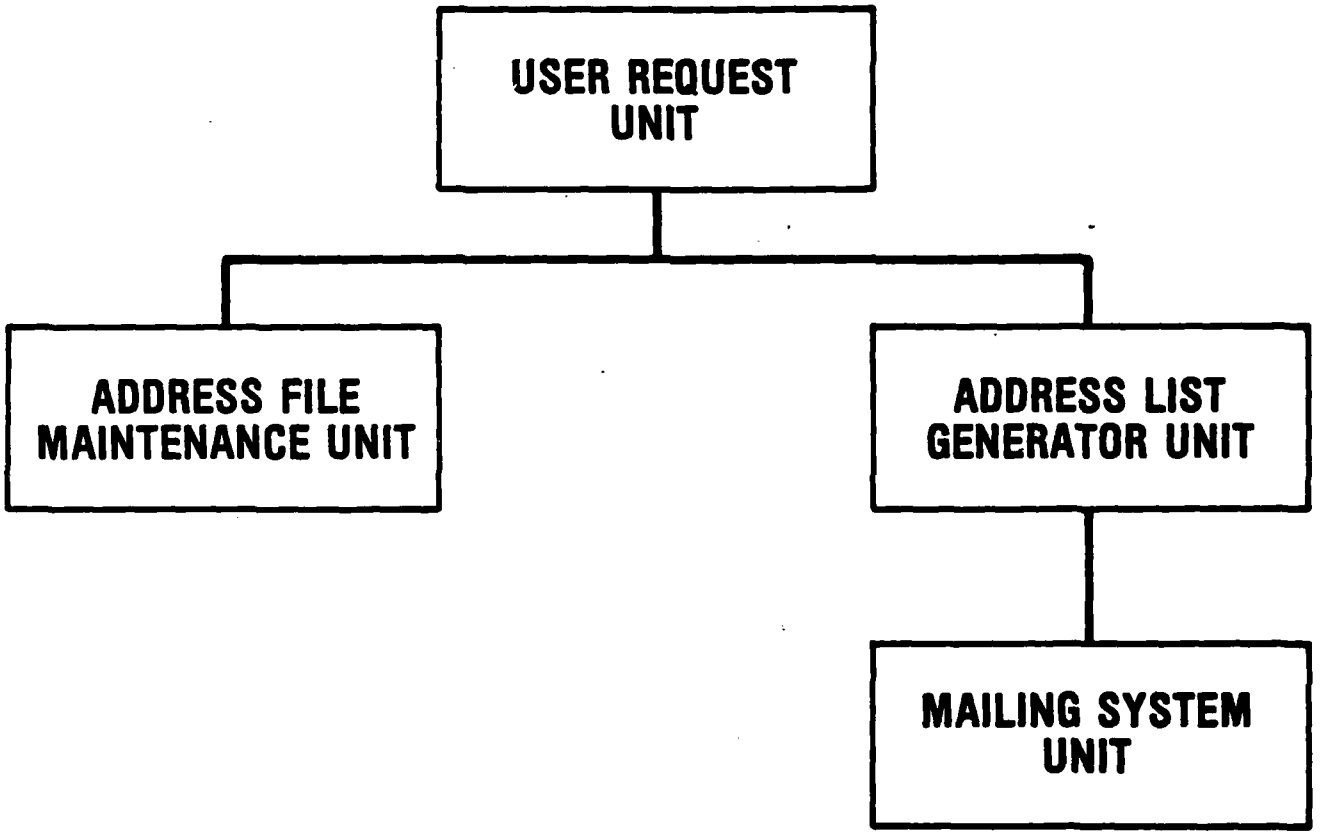


FIGURE 1. FUNCTIONAL DECOMPOSITION METHODOLOGY SAMPLE PROGRAM STRUCTURE

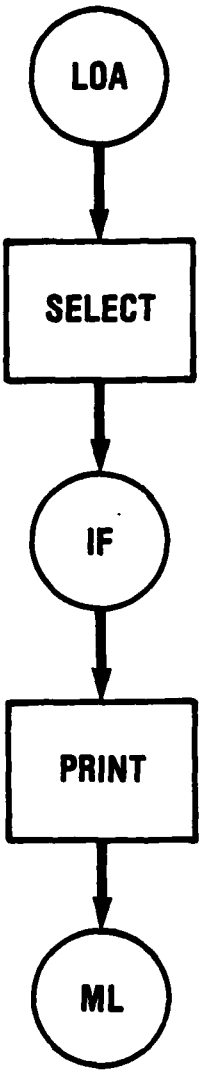


FIGURE 2. JACKSON PROGRAM DESIGN METHODOLOGY SAMPLE PROGRAM STRUCTURE

ARCHITECTURE:

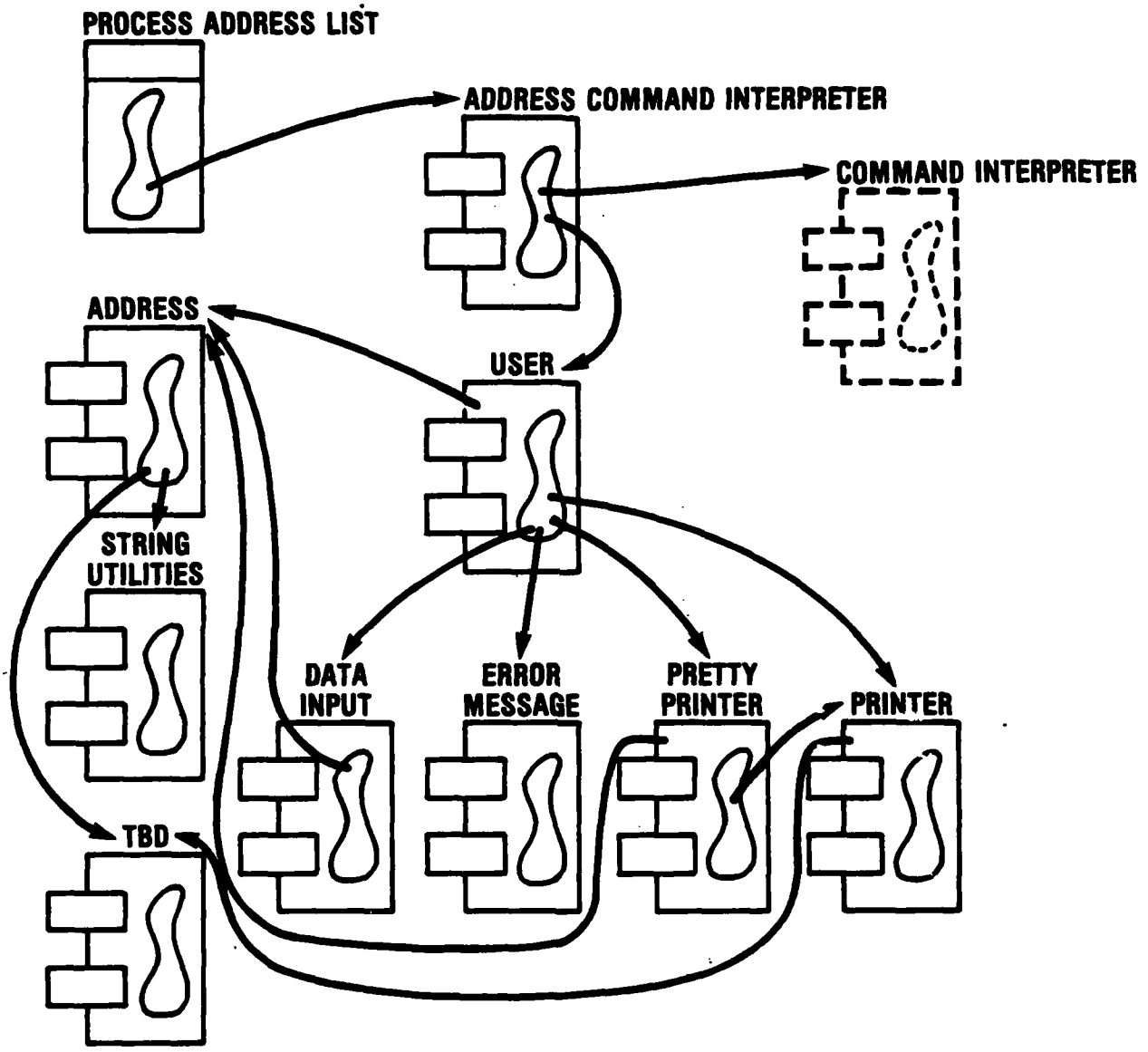


FIGURE 3. OBJECT-ORIENTED METHODOLOGY
SAMPLE PROGRAM STRUCTURE

Consistency

The definition of a program design methodology also implies that designs generated by different programmers using the same methodology should be similar in nature. While it is recognized that the basic design may become corrupted to some extent by later considerations (such as efficiency), the original designs should be consistent from programmer to programmer.

In this project, an attempt was made to generate measures of the consistency of the problem solutions, where consistency was defined as the amount of overlap in the structure of the programs. The intent was to index the proportion of functionally equivalent modules and functionally equivalent processing within modules for each program design methodology. Due to the different approaches taken to decomposition within each methodology, the definition of consistency within each decomposition technique would differ slightly.

For example, for object-oriented design, consistency would be measured in terms of the number of common objects and the amount of common processing on those objects. For functional decomposition, consistency would be measured in terms of the number of common processing steps, as defined by the modularization. Finally, for the Jackson program design methodology, consistency would be measured as the proportion of common data structures and proportion of common operations on each data structure.

There were large differences in the level of detail for each of the solutions provided. In addition, many of the rough drafts lacked a well-defined overall structure of the solution. This made it impossible to calculate a stable, objective measure of solution consistency.

DISCUSSION

The data provided a number of interesting insights into the implications of producing software using different design techniques. It must be made clear at the outset, however, that the data did not provide any clear answers regarding their relationship to future maintainability of the software produced.

The completeness, complexity, and design time data would seem to suggest that there is an advantage to generating program solutions using the Jackson program design and object-oriented methodologies. The solutions generated by these methodologies were more complete than other solutions. This is especially interesting in light of the fact that there was no significant difference in the amount of time that the programmers worked to produce these solutions. This suggests that completeness was not just a function of the amount of time spent working on the problem. Further, an examination of the weighted design times suggests that the Jackson and object-oriented approaches require less time to design and code the solution.

The complexity ratings again favor the Jackson and object-oriented methodologies, as they show lower complexity ratings than the other solutions. Low complexity ratings are important since they are argued to be the key to more maintainable code. They are especially important in this case, where the less complex solutions were completed in the least amount of time. Prior to the experiment, it was hypothesized that it would take longer to produce less complex code. The results here suggest cost savings not only in the maintenance phase, but also in the design phase.

We must be cautious, however, in our generalizations. It is possible that something other than the program design methodologies was responsible for the observed differences. The programmers using the Jackson program design methodology had a great deal more experience in programming, and with the design methodology, than did the programmers using either the object-oriented or functional decomposition approach. This might suggest that experience alone could account for the observed results. Other considerations, however, suggest that this may not be the case. The

difference in programming experience between the programmers using the Jackson and object-oriented techniques is very large (approximately 10 years) and yet there is only a 10% difference in the average completeness of the solutions. Likewise, there is only a small difference (approximately 1.5 years) between the programmers using the object and functional decomposition techniques and yet there is a 29% difference in the average completeness of their solutions. This suggests that *something* more than programmer experience is contributing to the observed differences in completeness.

Another factor suggesting that the program design methodologies may be the major contributor to the observed differences comes from the subjective evaluations of the structure of the obtained solutions. If experience is the major contributor to completion of the design, then it might be expected that the solutions generated by the more experienced programmers would be more complete and more similar, regardless of the program design methodology they used to produce the design. This did not appear to be the case in this research project. There was no correlation between percent complete and years of programming experience, and the *solutions generated* by the more experienced programmers were no more alike than the programs generated by the less experienced programmers.

In addition, the data suggested that certain of the methodologies were more appropriate than others for particular problems. For example, the Jackson program design methodology seemed most useful in solving the military address problem, which was a data-oriented problem. On the other hand, the object-oriented methodology seemed to produce better solutions for the host-at-sea buoy problem, which involved real-time processing. This also suggests that the methodologies were the contributing factor to the observed differences among solutions.

Overall, the data suggest that the different methodologies examined in this research do produce different solutions to the same set of problems. Further, they suggest that the methodologies do provide guidelines to programmers and that these guidelines may lead to reductions in design time and complexity of the design solution, at least for certain classes of

problems. Finally, the data suggest most strongly that further research is required — first, to disentangle the effects of program design methodology from experience, and second, to demonstrate the relationship between complexity and maintenance of programs.

ACKNOWLEDGEMENTS

The authors would like to thank Roger Baldwin of BPD of Union Carbide, Grady Booch of Rational Corporation, John Cameron of Michael Jackson Associates, Steve Ferg of the Federal Reserve Board, Gene Lowrimore of Decision Systems, Inc., Hans Nageli of Migros-Genossenschafts-Bund, Informatik, and Saul Portner of the General Electric Company, who were instrumental in helping us obtain participants for this research. We would also like to thank Dr. John J. O'Hare for his advice on this research.

REFERENCES

Boehm, B. W. (1981). Software Engineering Economics. Prentice-Hall, Inc.: Englewood Cliffs, N.J.

Boehm-Davis, D. A., Sheppard, S. B., Bailey, J. W. & Kruesi, E. An empirical evaluation of language-tailored PDLs (Tech. Rep. TR-82-388200-6), Arlington, Virginia: General Electric Company, 1981.

Curtis, B. & Sheppard, S. B. Identification and validation of quantitative measures of the psychological complexity of software (Tech. Rep. TR-79-388100-7), Arlington, Virginia: General Electric Company, 1979.

McCabe, T.J. A complexity measure. IEEE Transactions on Software Engineering, 1976, 2, 308-320.

Putnam, L. H. Measurement data to support sizing, estimating, and control of the software life cycle. In Proceedings of COMPCON '78. New York: IEEE, 1978.

Sheppard, S. B. & Kruesi, E. The effects of the symbology and spatial arrangement of software specifications in a coding task. In Proceedings of Trends & Applications 1981: Advances in Software Technology. New York: IEEE, 1981.

Sheppard, S.B., Kruesi, E., & Curtis, B. The effects of symbology and spatial arrangement on the comprehension of software specifications (Tech. Rep. TR-80-288200-2), Arlington, Virginia: General Electric Company, 1980.

Sheppard, S.B., Bailey, J.W., & Kruesi, E. The effects of the symbology and spatial arrangement of software specifications in a debugging task. (Tech. Rep. TR-81-388200-4), Arlington, Virginia: General Electric Company, 1981. (a)

Sheppard, S.B., Bailey, J.W., & Kruesi, E. The effects of the symbology and spatial arrangement of software documentation in a modification task. (Tech. Rep. TR-81-388200-5), Arlington, Virginia: General Electric Company, 1981. (b)

APPENDIX

Naval Air Station (NAS) Problem

Requirements

Introduction

The Navy wishes to determine the number of personnel using the facilities at NAS Emerald Point at any given time to ensure the most efficient utilization of the support facilities: housing (on- and off-base); eating facilities for Officers, Chief Petty Officers, enlisted and contracted civilian personnel; recreation; and medical facilities.

The problem is that in addition to the permanent staff, the base supports active squadrons, a Refresher Air Group (RAG), and a tenant training command. While the permanent staff is always in residence on the base, the number of people in residence from the other three groups will vary. Dates of arrival and departure for permanent and non-permanent residents of the base will be maintained in the data base (date of departure will be null for permanent residents).

Hardware

The base has a main computer facility with terminals in each of the squadron personnel offices as well as one for the RAG and tenant training command. The computer contains a data base (PERSONNEL_FILE) which allows operations of the following structure to be performed on it:

```
procedure RESET; [which sets the ID number to the first ID in the data
base]

procedure READ (ID: INTEGER); [which reads the next item in the data base
and returns the integer value of the next ID]

function END_OF_FILE return BOOLEAN; [which returns "true" when the end of
file is read]

procedure ARRIVAL_DATE(ID: INTEGER; D: DATE); [which takes as input the
value of the ID of the item to be read and returns the arrival date, as an
integer]

function ARRIVAL_DATE(ID: INTEGER) return DATE; [which takes as input the
value of the ID of the item to be read and returns the arrival date, as an
integer]

procedure NUM_DEPENDENTS (ID: INTEGER; D: NUM_DEPENDENTS); [which takes as
input the value of the ID of the item to be read and returns the number of
dependents, as an integer]

function NUM_DEPENDENTS(ID: INTEGER) return NUM_DEPENDENTS; [which takes
as input the value of the ID of the item to be read and returns the number
of dependents, as an integer]

                .
                .
                .
              etc.
```

Note: We have provided the same operation as both a procedure and a function; you may use whichever you feel more comfortable with. You can construct other needed operations by following the structure suggested by these examples.

Software Functions

The system software functions should include:

1. a data base to store the following information for interactive query:

   a. sailor ID
   b. a sailor's arrival date,
   c. a sailor's departure date,
   d. number of dependents,
   e. flag denoting temporary or permanent assignment to the base,
   f. rank, and
   g. civilian/military status flag.

   Note: New data will be made available weekly.

2. a utilization analysis for each facility (housing, eating, recreation, and medical). This analysis will identify the number of permanent and temporary staff who will use each of the facilities at any given point in time based on the requirements of the facility.

3. Hard copy output of the results of these utilization analyses will be required via a terminal and printer at the base personnel office.

Facility Requirements

Housing. To be eligible for housing, a sailor (officer or enlisted) must have 3 dependents and be ordered to the NAS for a minimum of 1 year.) The housing requirement is equal to the number of people meeting these requirements.

Eating. Civil servants and contracted personnel may only use the mess hall for the noon meal. All servicemen and retired personnel may use the eating facilities at any time. It is assumed that 70% of active duty personnel and 5% of the retired community will use the eating facility at any given time.

Recreation. It is assumed that 50% of the people on the base will use the recreational facilities at any given time. Recreation facilities may be used by all base support personnel, all other personnel assigned to the base, all beach squadrons whose complement is stationed at the base, and all of the retired community.

Medical. It is assumed that 30% of the people on the base will require medical attention at any given time. Medical attention is allowed for all base support personnel, all other personnel assigned to the base, all beach squadrons whose complement is stationed at the base, and all of the retired community.

The Host-at-Sea (HAS) Buoy System

Requirements

Introduction

The Navy intends to deploy HAS buoys to provide navigation and weather data to air and ship traffic at sea. The buoys will collect wind, temperature and location data, and will broadcast summaries periodically. Passing vessels will be able to request more detailed information. In addition, HAS buoys will be deployed in the event of accidents at sea to aid sea search operations.

Rapid deployment and the use of disposable equipment are novel features of HAS. HAS buoys will be relatively inexpensive, lightweight systems that may be deployed by being dropped from low-flying aircraft. It is expected that many of the HAS buoys will disappear because of equipment deterioration, bad weather conditions, accidents, or hostile action. The ability to redeploy rather than to attempt to prevent such loss is the key to success in the HAS program. In this sense, HAS buoys will be disposable equipment. To keep costs down, government surplus components will be used as much as possible.

Hardware

Each HAS buoy will contain a small computer, a set of wind and temperature sensors, and a radio receiver and transmitter. Eventually, a variety of special purpose HAS buoys may be configured with different types of sensors, such as wave spectra sensors. Although these will not be covered by the initial procurement, provision for future expansion is required.

A computer has been chosen for the HAS buoy program. There are more than 3000 available as government-surplus equipment. The computer has been found suitable for the new HAS program by virtue of its low weight, low cost, and low power consumption. A preliminary study shows that the capacity of a single computer will be insufficient for some HAS configurations, but it has been decided to use two or more computers in these cases. Therefore, provision for multi-processing is required in the software.

Input is performed by using the procedure SENSE (procedure SENSE (D: Device) return CONTENTS) that selects a device and returns its contents ("instantaneously") to the main procedure. So, for example, the instruction SENSE Omega would return the Omega coordinates of the location.

The temperature sensors take air and water temperature (Centigrade). On some HAS buoys, an array of sensors on a cable will be used to take water temperature at various depths. The buoys are capable of sensing up to 20 temperatures, which they send back in one return when requested.

Because the surplus temperature sensors selected for HAS are not designed for sea-surface conditions, the error range on individual readings may be large. Preliminary experiments indicate that the temperature can be measured within an acceptable tolerance by averaging several readings from the same device. To improve the accuracy further and to guard against sensor failure, most HAS buoys have multiple temperature sensors (maximum = 5).

Each buoy will have one or more wind sensors (maximum = 3) to observe wind magnitude in knots and wind direction. Surplus propellor-type sensors have been selected because they meet power restrictions.

Buoy geographic position is determined by use of a radio receiver link with the Omega navigation system.

Some HAS buoys are also equipped with a red light and an emergency switch. The red light may be made to flash by a request radioed from a vessel during a sea-search operation. If the sailors are able to reach the buoy, they may flip the emergency switch to initiate SOS broadcasts from the buoy.

Software Functions

The software for the HAS buoy must carry out the following functions:

1. Maintain current wind and temperature information by monitoring sensors regularly and averaging readings.

2. Calculate location via the Omega navigation system.

3. Broadcast wind and temperature information every 60 seconds. Each broadcast takes 5 seconds.

4. Broadcast more detailed reports in response to requests from passing vessels. The information broadcast and the data rate will depend on the type of vessel making the request (ship or airplane).

5. Broadcast weather history information in response to requests from passing vessels. The history report consists of the every-60-second reports from the last 48 hours.

6. Broadcast an SOS signal in place of the ordinary every-60-second message after a sailor flips the emergency switch. This signal takes 2 seconds and should continue until a vessel sends a reset signal.

Software Timing Requirements

In order to maintain accurate information, readings must be taken from the sensing devices at the following fixed intervals:

temperature sensors:	every 10 seconds
wind sensors:	every 30 seconds
Omega signals:	every 10 seconds.

Since the buoy can only transmit one report at a time, conflicts will arise. If the transmitter is free and more than one report is ready, the next report will be chosen according to the following priority ranking:

SOS	1	highest
Airplane Request	2	
Ship/Satellite Request	3	
Periodic	4	lowest.

The Military Address System (MADDS)

Requirements

Introduction

Many organizations maintain lists of names and postal addresses in a computer. In simple applications, the whole list is used to generate a set of mailing labels or "personalized" letters. In other applications, a subset of the list is selected according to criteria believed to identify individuals most likely to be interested in the contents of the mailing. For example, a publisher who wished to offer a new magazine called Tax Loopholes might want to select addresses for people with medical degrees. Others might want to select all persons within a particular geographic area (consider a magazine like Southern Living, for example), while still others might be interested in persons with specific first or last names.

The address lists can be obtained from various sources, such as magazine subscription departments, and are generally delivered on a medium such as magnetic tape. Data from different sources are likely to appear in different record formats.

The task for any software system that processes such a list is to read the input data (in a specified, but likely-to-change, format), extract the desired subset of the list according to certain criteria (which are specified, but likely to change), and print that subset (also in a specified, but likely-to-change, format).

Software Functions

The system should be able to search for and print the addresses within a certain Postal code area, and/or to do the same for the addresses with a certain O-grade. (An O-grade is a numerical representation of an officer's rank.) MADDS has three external interfaces: the user, the address list generator, and the mailing system.

A request to run the application does not specify the order in which to create the output. That is, it does not request that the O-grade application be run first, and then the Postal code application, or vice versa. Nor does it imply that the two applications are in fact separate tasks.

Mailing List Input Format

The list of addresses input to MADDS can be read in character by character. Addresses follow directly one after another, each in a separate record, and can be read in as separate records. Each address consists of 11 fields. The fields follow directly after one another as specified in the following table.

Field	Name	Field Size (Number of Characters)	Content[1]
1	Title	4	E.g., "Mr.", "Ms.", "Dr.", "Capt."[2]
2	Last Name	15	
3	Given Names	20	Two strings separated by at least one blank; First string is first name, second is middle[3]
4	Branch or Code	20	
5	Command or Activity	20	
6	Street or P.O. Box	20	
7	City	20	
8	State or province	20	
9	Country	15	
10	Postal Code	10	Contiguous letters or digits
11	O-grade	4	

---

1. The value of any field can be an all-blank string. If the value of a field is not an all-blank string, then its first character will be non-blank (i.e., values are left justified). Any example field value shorter than the field size should be considered to be padded on the right with blanks.

2. The last address of the file is fake, having only a title field consisting completely of asterisks. It is an end-of-file marker.

3. If there is no middle initial, this will be signalled by a special character in the record which should be printed as a blank space when the address is output.

As mentioned in the introduction, it is possible that the format of the address file might change. For example, the 11 fields might follow one another in an order different than the one specified above, or the number of fields might change.

Mailing List Output Format

Each address produced by MADDS should adhere to the following format:

Line 1: Title
single blank
Given Names
single blank
Last Name

Line 2: Branch or Code

Line 3: Command or Activity

Line 4: City
comma
single blank
State or Province

Line 5: Country
single blank
Postal Code

Addresses are separated from one another by five blank lines. Each output line is to begin in the first column.

As mentioned in the introduction, the output format could also change. The new format might output the information in a different order, or it might output a different subset of the information (from one piece of data to all contained in the record).

TECHNICAL REPORTS DISTRIBUTION LIST

OSD

CAPT Paul R. Chatelier
Office of the Deputy Under Secretary of Defense
OUSDRE (E&LS)
Pentagon, Room 3D129
Washington, D. C. 20301

Department of the Navy

Engineering Psychology Program
Office of Naval Research
Code 442EP
800 N. Quincy Street
Arlington, VA 22217 (3 cys.)

Manpower, Personnel & Training Programs
Code 270
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Information Sciences Division
Code 433
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Special Assistant for Marine Corps Matters
Code 100M
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Director
Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D. C. 20375

Dr. Michael Melich
Communications Sciences Division
Code 7500
Naval Research Laboratory
Washington, D. C. 23075

Department of the Navy cont'd

Dr. J. S. Lawson
Naval Electronic Systems Command
NELEX-06T
Washington, D. C. 20360

Naval Training Equipment Center
ATTN: Technical Library
Orlando, FL 32813

Dr. Robert G. Smith
Office of the Chief of Naval Operations, OP987H
Personnel Logistics Plans
Washington, D. C. 20350

Human Factors Department
Code N-71
Naval Training Equipment Center
Orlando, FL 32813

Human Factors Engineering
Code 8231
Naval Ocean Systems Center
San Diego, CA 92152

Dean of Research Administration
Naval Postgraduate School
Monterey, CA 93940

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
Code RD-1
Washington, D. C. 20380

Dr. L. Chmura
Naval Research Laboratory
Code 7592
Computer Sciences & Systems
Washington, D. C. 20375

Professor Douglas E. Hunter
Defense Intelligence College
Washington, D. C. 20374

Department of the Navy cont'd

CDR C. Hutchins
Code 55
Naval Postgraduate School
Monterey, CA 93940

Human Factors Technology Administrator
Office of Naval Technology
Code MAT 0722
800 N. Quincy Street
Arlington, VA 22217

CDR Tom Jones
Naval Air Systems Command
Human Factors Programs
NAVAIR 330J
Washington, D. C. 20361

Commander
Naval Air Systems Command
Crew Station Design
NAVAIR 5313
Washington, D. C. 20361

Mr. Philip Andrews
Naval Sea Systems Command
NAVSEA 61R
Washington, D. C. 20362

Commander
Naval Electronics Systems Command
Human Factors Engineering Branch
Code 81323
Washington, D. C. 20360

Dr. George Moeller
Human Factors Engineering Branch
Submarine Medical Research Lab
Naval Submarine Base
Groton, CT 06340

Dr. Robert Blanchard
Navy Personnel Research and
Development Center
Command and Support Systems
San Diego, CA 92152

Mr. Stephen Merriman
Human Factors Engineering Division
Naval Air Development Center
Warminster, PA 18974

Human Factors Engineering Branch
Code 4023
Pacific Missile Test Center
Point Mugu, CA 93042

Department of the Navy cont'd

Dean of the Academic Departments
U. S. Naval Academy
Annapolis, MD 21402

Department of the Army

Dr. Edgar M. Johnson
Technical Director
U. S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Technical Director
U. S. Army Human Engineering Labs
Aberdeen Proving Ground, MD 21005

Director, Organizations and
Systems Research Laboratory
U. S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Mr. J. Barber
HQS, Department of the Army
DAPE-MBR
Washington, D. C. 20310

Department of the Air Force

Dr. Kenneth R. Boff
AF AMRL/HE
Wright-Patterson AFB, OH 45433

U. S. Air Force Office of
Scientific Research
Life Science Directorate, NL
Bolling Air Force Base
Washington, D. C. 20332

AFHRL/LRS TDC
Attn: Susan Ewing
Wright-Patterson AFB, OH 45433

Chief, Systems Engineering Branch
Human Engineering Division
USAF AMRL/HES
Wright-Patterson AFB, OH 45433

Dr. Earl Alluisi
Chief Scientist
AFHRL/CCN
Brooks Air Force Base, TX 78235

Other Government Agencies

Defense Technical Information Center
Cameron Station, Bldg. 5
Alexandria, VA 22314 (12 copies)

Dr. Clinton Kelly
Defense Advanced Research Projects
Agency
1400 Wilson Boulevard
Arlington, VA 22209

Other Organizations

Dr. Jesse Orlansky
Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria, VA 22043

Dr. Paul E. Lehner
PAR Technology Corporation
Seneca Plaza, Route 5
New Hartford, N.Y. 13413

Dr. Stanley Deutsch
NAS-National Research Council (COHF)
2101 Constitution Avenue, N.W.
Washington, D. C. 20418

Mr. Edward M. Connelly
Performance Measurement Associates, Inc.
1909 Hull Road
Vienna, VA 22180

National Security Agency
ATTN: N-32, Marie Goldberg
9800 Savage Road
Ft. Meade, MD 20722

Dr. Marvin Cohen
Decision Science Consortium, Inc.
Suite 721
7700 Leesburg Pike
Falls Church, VA 22043

Dr. Richard Pew
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02238

Dr. Douglas Towne
University of Southern California
Behavioral Technology Laboratories
1845 South Elena Avenue, Fourth Floor
Redondo Beach, CA 90277

END

FILMED

1-85

DTIC