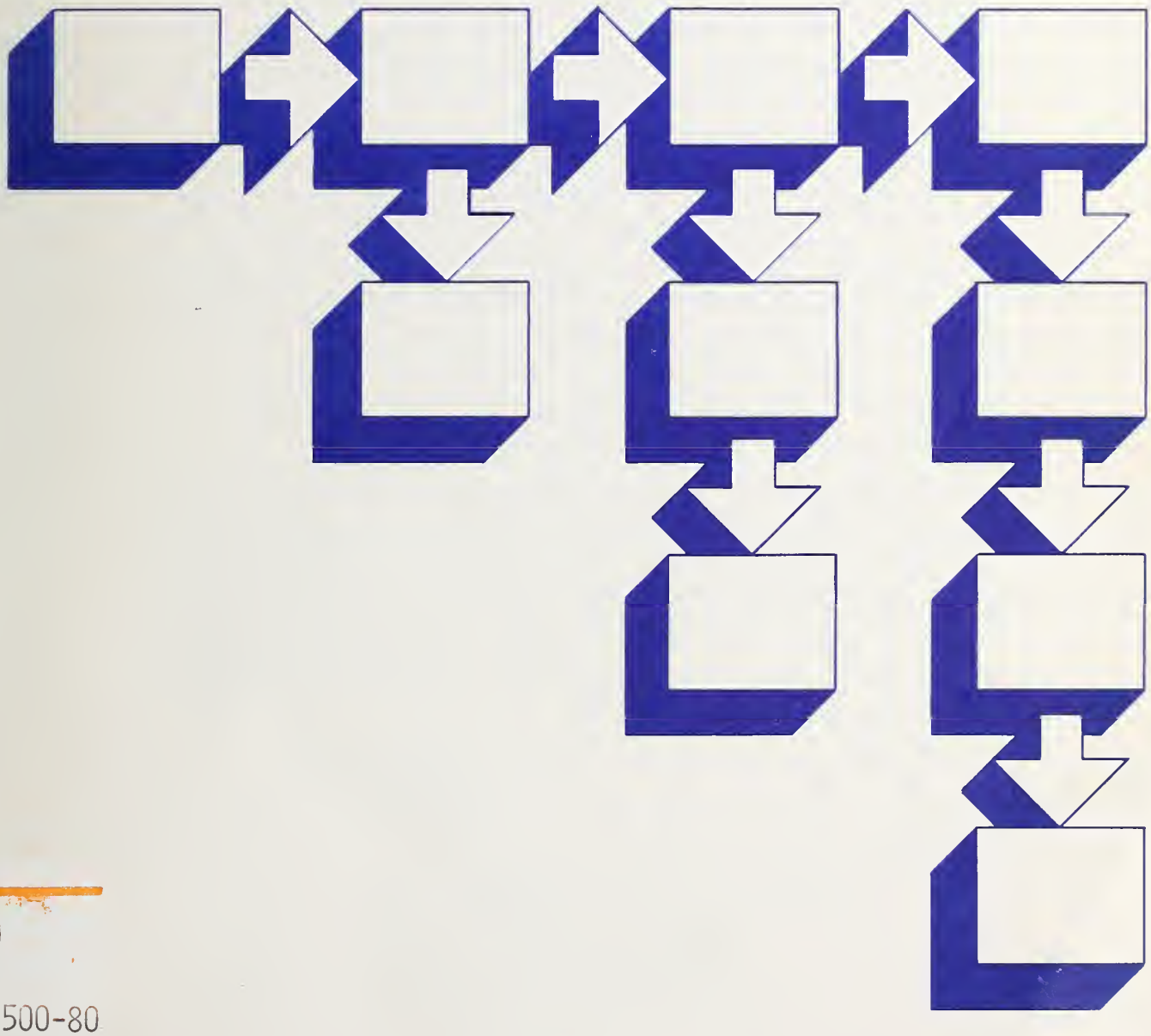
U.S. Department
of Commerce

National Bureau
of Standards

Computer Science and Technology

NBS Special Publication 500-80

Proceedings of the NBS/IEEE/ACM Software Tool Fair



NATIONAL BUREAU OF STANDARDS

The National Bureau of Standards[1] was established by an act of Congress on March 3, 1901. The Bureau's overall goal is to strengthen and advance the Nation's science and technology and facilitate their effective application for public benefit. To this end, the Bureau conducts research and provides: (1) a basis for the Nation's physical measurement system, (2) scientific and technological services for industry and government, (3) a technical basis for equity in trade, and (4) technical services to promote public safety. The Bureau's technical work is performed by the National Measurement Laboratory, the National Engineering Laboratory, and the Institute for Computer Sciences and Technology.

THE NATIONAL MEASUREMENT LABORATORY provides the national system of physical and chemical and materials measurement; coordinates the system with measurement systems of other nations and furnishes essential services leading to accurate and uniform physical and chemical measurement throughout the Nation's scientific community, industry, and commerce; conducts materials research leading to improved methods of measurement, standards, and data on the properties of materials needed by industry, commerce, educational institutions, and Government; provides advisory and research services to other Government agencies; develops, produces, and distributes Standard Reference Materials; and provides calibration services. The Laboratory consists of the following centers:

Absolute Physical Quantities[2] — Radiation Research — Thermodynamics and Molecular Science — Analytical Chemistry — Materials Science.

THE NATIONAL ENGINEERING LABORATORY provides technology and technical services to the public and private sectors to address national needs and to solve national problems; conducts research in engineering and applied science in support of these efforts; builds and maintains competence in the necessary disciplines required to carry out this research and technical service; develops engineering data and measurement capabilities; provides engineering measurement traceability services; develops test methods and proposes engineering standards and code changes; develops and proposes new engineering practices; and develops and improves mechanisms to transfer results of its research to the ultimate user. The Laboratory consists of the following centers:

Applied Mathematics — Electronics and Electrical Engineering[2] — Mechanical Engineering and Process Technology[2] — Building Technology — Fire Research — Consumer Product Technology — Field Methods.

THE INSTITUTE FOR COMPUTER SCIENCES AND TECHNOLOGY conducts research and provides scientific and technical services to aid Federal agencies in the selection, acquisition, application, and use of computer technology to improve effectiveness and economy in Government operations in accordance with Public Law 89-306 (40 U.S.C. 759), relevant Executive Orders, and other directives; carries out this mission by managing the Federal Information Processing Standards Program, developing Federal ADP standards guidelines, and managing Federal participation in ADP voluntary standardization activities; provides scientific and technological advisory services and assistance to Federal agencies; and provides the technical foundation for computer-related policies of the Federal Government. The Institute consists of the following centers:

Programming Science and Technology — Computer Systems Engineering.

[1]Headquarters and Laboratories at Gaithersburg, MD, unless otherwise noted; mailing address Washington, DC 20234.
[2]Some divisions within the center are located at Boulder, CO 80303.

Computer Science and Technology

NBS Special Publication 500-80

Proceedings of the NBS/IEEE/ACM Software Tool Fair

Held in conjunction with the
5th International Conference on Software Engineering
in San Diego, CA, March 10-12, 1981

Editor:

Raymond C. Houghton, Jr.

Center for Programming Science and Technology
Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, DC 20234

Sponsored by:

National Bureau of Standards
Washington, DC 20234

and

IEEE Computer Society
P.O. Box 639
Silver Spring, MD 20901

and

SIGSOFT ACM
1133 Avenue of the Americas
New York, NY 10036

U.S. DEPARTMENT OF COMMERCE
Malcolm Baldrige, Secretary

National Bureau of Standards
Ernest Ambler, Director

Issued October 1981

Reports on Computer Science and Technology

The National Bureau of Standards has a special responsibility within the Federal Government for computer science and technology activities. The programs of the NBS Institute for Computer Sciences and Technology are designed to provide ADP standards, guidelines, and technical advisory services to improve the effectiveness of computer utilization in the Federal sector, and to perform appropriate research and development efforts as foundation for such activities and programs. This publication series will report these NBS efforts to the Federal computer community as well as to interested specialists in the academic and private sectors. Those wishing to receive notices of publications in this series should complete and return the form at the end of this publication.

National Bureau of Standards Special Publication 500-80
Nat. Bur. Stand. (U.S.), Spec. Publ. 500-80, 238 pages (Oct. 1981)
CODEN: XNBSAV

Library of Congress Catalog Card Number: 81-600109

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1981

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402
Price $6.50
(Add 25 percent for other than U.S. mailing)

FOREWORD

The Institute for Computer Sciences and Technology (ICST) within the National Bureau of Standards (NBS) has a mission under Public Law 89-306 (Brooks Act) to develop standards to enable the "economic and efficient purchase, lease, maintenance, operation, and utilization of automatic data processing equipment by Federal Departments and Agencies." As part of its current standards initiative, ICST is studying, evaluating, and publishing methods that increase the productivity and quality of software procured by the Government and software developed within the Government. A way to achieve higher quality and increased programmer productivity appears to lie in the use of computer technology itself. Automation must be used to serve and augment itself. A recent GAO report [1] endorses this use of automation and concludes that software tools can offer the Federal Government the following:

-- Better management control of computer software development, operation, maintenance, and conversion.

-- Lower costs for computer software development, operation, maintenance, and conversion.

-- Feasible means of inspecting both contractor-developed and in-house-developed computer software for such quality indications as conformance to standards and thoroughness of testing.

To this end, ICST has established a project to study, evaluate, and make recommendations concerning the use of software development tools. A database on existing tools has been assembled and is available in published form [2]. A taxonomy of software tool features has been produced [3] to provide a means for classifying and evaluating the capabilities of the ever more complex tools available in the marketplace. As a part of the program to provide information to Federal Agencies on tool usage, capabilities, limitations, and availability, NBS co-sponsored the tool fair held in conjunction with the Fifth International Conference on Software Engineering. The concept for the tool fair came from Professor Gerald Estrin, Chairperson, Computer Science Department, UCLA, as a result of experiences with remote interactive demonstrations of SARA (System ARchitects Apprentice) over a 3 year period.

The proceedings of the tool fair are intended for use by all Federal and private organizations seeking information about current and future software engineering tools. The views expressed in the papers do not necessarily reflect those of the National Bureau of Standards, and the inclusion of any given tool, whether or not commercially available, does not imply that NBS endorses or warrants its operability or suitability.

---

[1] "Wider Use of Better Computer Software Technology Can Improve Management Control and Reduce Costs", FGMSD-80-38, Apr 80.
[2] "NBS Software Tools Database", NBSIR 80-2159, Oct 80.
[3] "Features of Software Development Tools", NBS SP 500-74, Feb 81.

PREFACE

The San Diego Tool Fair was a first-of-its-kind demonstration of software engineering tools at a major conference. The goals of the tool fair were to preview tools currently in research, to increase knowledge of what available tools do, and to demonstrate how tools are used in software engineering. However, a more important goal of the tool fair was to make the demonstrations an extension of the technical proceedings of the conference. This goal was accomplished by emphasizing the technical aspects of the tools in a professional, non-sales manner.

The tool fair ran parallel with the conference program. Demonstrations were held in private conference rooms and were scheduled on a periodic basis. Over 33 tools were demonstrated at the tool fair and included the following:

Requirements, Design, and Modeling Tools
Cost Estimation and Project Management Tools
Program Analysis and Testing Tools
Configuration Management Systems
Formal Verification Systems
Program Construction and Generation Systems
Programming Environments

The majority of the tools demonstrated are not marketed commercially. Many of these tools were developed by universities or Government-supported research organizations, while many others were developed for internal use by organizations.

This document summarizes the presentations made by each demonstrator at the San Diego Tool Fair. These summaries are arranged in chronological and station order, so they reflect the published schedule of the tool fair. Each summary includes a short description of the tool, a scenario of the demonstration, a list of references, background on the demonstrators, sample output, and a page of miscellaneous data obtained from the NBS Software Tools Database (see page 180). The appendix provides a cross reference to the features of the tools and can be used as an index to identify tools of interest.

The organizing committee would like to acknowledge the support of the following people and thank them for their contributions: Mr. Seymour Jeffery, General Chairperson, and Dr. Leon G. Stucki, Program Chairperson of ICSE5, for their foresight and support of Tool Fair; Mr. Robert Fritz of General Dynamics Corporation, San Diego, for his support in finding resources for many of the demonstrations; Dr. Martha Branstad, NBS, and Ms. Patricia Powell, NBS, for their helpful guidance and on-site assistance; and finally, the Review Committee for their support in reviewing the many proposals received for Tool Fair.

Dr. Gerald Estrin, Chairperson
Mr. Raymond C. Houghton, Vice Chairperson
Tool Fair Organizing Committee

ORGANIZING COMMITTEE

G. Estrin, Chairperson R. Houghton, Vice Chairperson

REVIEW COMMITTEE

J. Barkley	G. Cannon, Jr.	R. Fairley
S. Frankel	H. Hecht	B. Leong-Hong
G. Lyon	E. Miller	K. Moore
A. Neumann	L. Osterweil	M. Penedo
P. Powell	R. Razouk	D. Reifer
W. Riddle	P. Santoni	M. Zelkowitz

ABSTRACT

This document summarizes the presentations made by each demonstrator at the San Diego Tool Fair. The San Diego Tool Fair was a first-of-its-kind demonstration of software engineering tools at a major conference. Each summary includes a short description of the tool, a scenario of the demonstration, a list of references, background on the demonstrators, sample output, and a page of miscellaneous data obtained from the NBS Software Tools Database. The appendix provides a cross reference to the features of the tools.

Key words: programming aids; software automation; software development; software engineering; software testing; software tools.

EDITOR'S COMMENT

Most of the tool descriptions and scenarios that appear in this publication were received directly from the authors/demonstrators and were reviewed by members of the review committee. In some cases, the text was modified for brevity or adherence to NBS publishing guidelines. Sample output was either directly received from the authors/demonstrators or acquired from handouts distributed at the conference. The Appendix and the lists of tool features were obtained from the NBS Software Tools Database. Definitions for most of these features can be found in NBS Letter Circular-1127, Software Development Tools: A Reference Guide to a Taxonomy of Tool Features, Feb. 81. Copies of this reference guide may be requested from: The Software Tools Project, Room A-265 Technology, National Bureau of Standards, Washington, DC 20234.

NOTE

In no case does identification of commercial products imply recommendation or endorsement by the National Bureau of Standards, nor does it imply that the product identified is necessarily the best available for the purpose.

TABLE OF CONTENTS and TOOL FAIR SCHEDULE

March 10, 1981

Station 1

Station 2

Station 3

Station 4

Station 5

---

* SOFTOOL 80 is a Trade Mark of Softool Corporation

---

* IFTRAN and RXVP80 are Trade Marks of General Research Corporation

CS4

A TOOL FOR THE DESIGN AND GENERATION OF DATABASE SYSTEMS

Stig Berild and Sam Nachmens

SYSLAB
University of Stockholm
S-106 91 Stockholm
Sweden

Databaskonsult DBK AB
Huvudstagatan 12
S-171 58 Solna
Sweden

1. INTRODUCTION

The CS4 system was originally designed by the CADIS Research Group at the Royal Institute of Technology and the University of Stockholm. Since 1978, CS4 has been maintained and has been further developed by Databaskonsult DBK AB, in parallel with further conceptual development within SYSLAB (the System Development Laboratory) at the University of Stockholm.

CS4 has been operational since 1975, and has been used for a variety of applications - from simple "telephone-list" systems to complex systems, e.g. for planning and resource allocation. CS4 has also been used for education at several universities in Europe - both as a first programming language and as a tool at more advanced courses on databases.

CS4 is available on DEC-10/DEC-20 under TOPS-10 or TOPS-20, and on UNIVAC-1100 under EXEC-8. An IBM/TSO version is under development (planned release during the summer of 1981), and an implementation on VAX is planned.

2. SUMMARY OF CS4

CS4 is a systems development package, developed mainly as a tool for quick and flexible implementation of prototype systems as well as minor production systems. CS4 includes an interpreter for an easily learned general purpose programmming language, specially designed for handling associative databases.

In a CS4 associative database, information is represented as associations between entities. This simple but powerful representation facilitates handling of data of arbitrary complexity.

Small, self-contained procedures are built up and stored in a procedure library. Procedures can call each other arbitrarily and recursively.

3. SCENARIO OF A CS4 DEMONSTRATION

A demonstration starts with a short description of the CS4 associative database - how data are represented and retrieved. This description takes only a few minutes, but is essential for an understanding of the demonstration.

The audience will then be asked to specify a small system, which is successively implemented and extended according to the specifications of the audience. The emphasis will be put on showing the flexibility of the associative database.

A demonstration will take about 30 minutes, whereafter 30 minutes will be set aside for those wishing to further extend the system being built by the audience, or have any other aspect of CS4 demonstrated.

4. CS4 LITERATURE

1. Janning, Berild, Nachmens, "Introduction to Associative Databases and the CS4 System", is a text book giving a comprehensive introduction to the main CS4 features. It can be read by persons without programming experience. Published by Studentlitteratur, Lund, Sweden, 1981. Available in Swedish and English.

2. Berild, Nachmens, "Some Practical Applications of CS4 - a DBMS for Associative Databases", in Nijssen, "Architecture and Models in Database Management Systems", North-Holland 1977, pp. 213-236.

3. Berild, Nachmens, "CS4 - A tool for Database Design by Infological Simulation", VLDB-3, Tokyo, 1977 (published in Ramamoorthy, Yeh, "Tutorial: Software Methodology", COMPSAC 78).

4. Nachmens, "Associative Databases for Changing Information Requirements", 13th Hawaiian International Conference on Systems Sciences, Honolulu, 1980.

5. THE DEMONSTRATORS

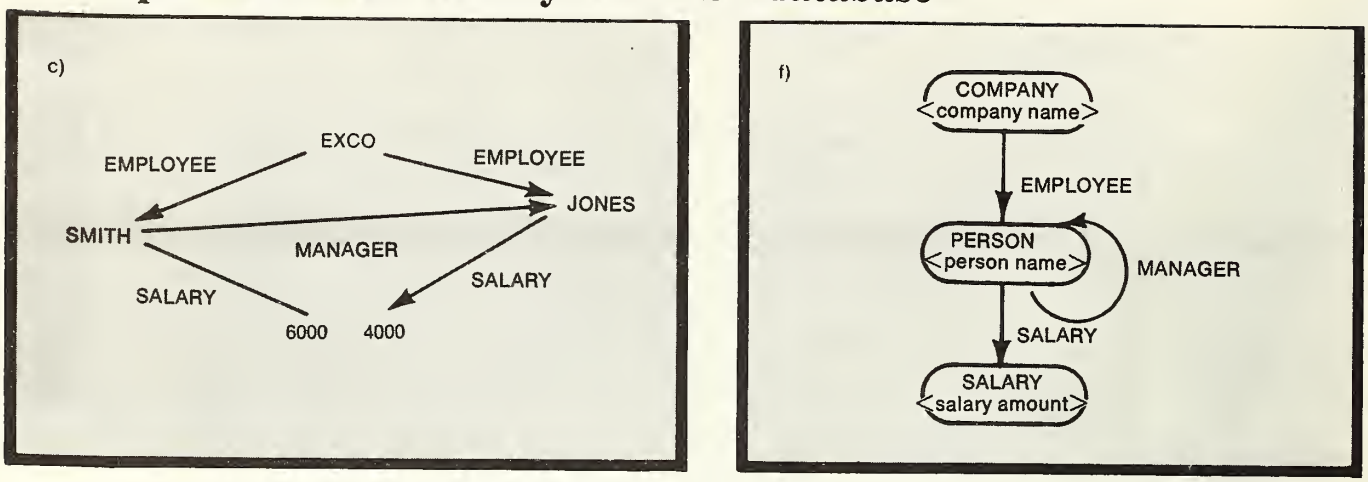
Stig Berild

Stig Berild has been a researcher at the Department of Computer Science, University of Stockholm, since 197Ø. He is currently a member of the Sytems Development Laboratory, working with database-oriented problems and with development of computer-aided tools for database design, with special emphasis on the further improvement of CS4. Since 1978, he is also connected to Databaskonsult DBK AB, which is a Swedish company that now is responsible for the product orientation of the CS4 system.

Sam Nachmens

Sam Nachmens has been working as a researcher at the Department of Computer Science, University of Stockholm, since 1972. He has been working with the development of the CS4 system, and with related information-structuring and data-structuring problems. Since 1978, he has been also working within Databaskonsult DBK AB.

Representation of reality in a CS4-database



Occurrence level

Conceptual level

Each association is represented by an e-record (elementary record) indicating the name of the association and the two entities involved. The representation in figure c can now be described by the following e-records:

```
('EMPLOYEE', 'EXCO', 'JONES')
('EMPLOYEE', 'EXCO', 'SMITH')
('MANAGER', 'SMITH', 'JONES')
('SALARY', 'SMITH', '6000')
('SALARY', 'JONES', '4000')
```

An e-record (A, B, C) is interpreted as "A of B is C"—for instance "EMPLOYEE of EXCO is JONES".

Retrieval of information

Those instructions in CSL which refer to information in the database use search templates to describe which e-records are to be retrieved. A search template has the same structure as an e-record, i.e. it has three components. Up to three components of a template can be specified (i.e. known)—they are names. Other components can be indicated as 'searched-for' (=) or 'irrelevant' (*). A search template is logically matched against all e-records of the database and those 'searched-for' names where the template's specified components match the corresponding ones of the e-records are retrieved. Some examples are:

('MANAGER', 'SMITH', 'JONES')	the value is TRUE, if such an e-record exists, else FALSE
('SALARY', 'SMITH', =)	retrieves salary amount(s) associated with SMITH
('MANAGER', =, 'JONES')	retrieves all persons which have JONES as MANAGER
('EMPLOYEE', *, =)	retrieves names of all persons which participate in some EMPLOYEE association
(=, 'JONES', *)	retrieves names of all associations which emanate from JONES.

Example 1

Assume a database with a structure according to figure f on the preceeding page. We now wish to specify a procedure which for each employee of EXCO lists his/her name and salary.

```
10  BEGIN
20  ENTITY EMP, SAL
30  FORMAT LINE=<1,15<EMP;>17,25>SAL
40  OUTPUT 'EMPLOYEE                SALARY', SPACE
50  FOREACH EMP IN ('EMPLOYEE', 'EXCO',=) DO
60    GET SAL IN ('SALARY', EMP,=)
70    OUTPUT LINE
80  NEXT
90  END
```

SAMPLE OUTPUT

A session for input and test of the procedure according to example 1 may look as follows:

```
.  R CS4                          Initiate CS4. Start of session
*  ASSIGN DB EXBAS                Assign a database
*  ASSIGN LIB EXLIB               Assign a procedure library
*  INSERT                         Command for automatic generation of line numbers
10 BEGIN                          }
.                                 }
.                                 }  Input of procedure
.                                 }
90 END                            }
*  NAME SALARYLIST                Assign name to procedure
*  SAVE                           Save procedure in library
*  RUN SALARYLIST                 Execute procedure
EMPLOYEE          SALARY          }
                                  }
ANDERSON            5000          }
ERICSON             5600          }
JONES               4000          }
.                                 }
.                                 }  The salary list is printed
.                                 }
SMITH               6000          }
.                                 }
.                                 }
.                                 }
ZIMMERMAN          11500          }
*  STOP                           End of CS4-session
```

Example 2

Assume a database with the same structure as before. We now wish to design a procedure which first requests a name of an employee of EXCO and then lists all his/her subordinates, the subordinates of the subordinates etc., according to the following lay-out:

WHICH EMPLOYEE: JONES

JONES
  ANDERSON
    CARLSON
    SMITH
  LEE
    DAVIDSON
    etc

In order to achive this we specify two procedures. We start with the procedure which queries the user for an employee name.

```
10  BEGIN
20  ENTITY EMP
30  QUERY 'WHICH EMPLOYEE: ' :EMP
40  IF ('EMPLOYEE', 'EXCO', EMP) EXIST
50    THEN SUBORDINATE EMP, 1
60    ELSE OUTPUT EMP; ' IS NOT EMPLOYED BY EXCO'
70  END
```

The SUBORDINATE procedure is defined as follows:

```
10  IN-ENTITY EMP, LEVEL
20  BEGIN
30  ENTITY SUB, LINE
40  LET LINE (LEVEL, LEVEL+15)=EMP
50  OUTPUT LINE
60  FOREACH SUB IN ('MANAGER', =, EMP) DO
70    SUBORDINATE SUB, LEVEL+2
80  NEXT
90  END
```

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . DATA INPUT
. . CODE INPUT
. . . CSL
FUNCTION
. TRANSFORMATION
. . TRANSLATION
. STATIC ANALYSIS
. . MANAGEMENT
. . . DATA BASE MANAGEMENT
OUTPUT
. USER OUTPUT
. . LISTINGS
. . USER-ORIENTED TEXT
. . . DOCUMENTATION
. MACHINE OUTPUT
. . OBJECT CODE OUTPUT
. . DATA OUTPUT

IMPLEMENTATION LANGUAGE: FORTRAN

TOOL PORTABLE: NO

COMPUTER (OTHER HARDWARE): IBM 360/370, DECSYSTEM 10/20, UNIVAC-1100, VAX

OS (OTHER SOFTWARE): TSO, EXEC-8, TOPS 10/20

TOOL AVAILABLE: YES

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): CONTACT DATABASKONSULT DBK AB

TOOL SUPPORTED: YES, TOOL SUPPORT: DATABASKONSULT DBK AB

CONTACT: SAM NACHMENS, DATABASKONSULT DBK AB, HUVUDSTAGATAN 12, SOLNA, S-171 58, SWEDEN, 08-83 07 30
STIG BERILD, UNIVERSITY OF STOCKHOLM, DEPT OF INFORMATION PROCESSING, FACK, STOCKHOLM, 10691, SWEDEN, 46-8-150160

LILITH - A MODULA MACHINE

1. Project Sponsor:

Institut fuer Informatik
Eidgenoessische Technische Hochschule
Zurich, Switzerland

2. Project Summary:

LILITH is a tool for the creation of programs and documents. It is a general purpose computer created to provide an optimal environment for the programming language MODULA II. The merit of this system as a program and document preparation tool rests upon the following features of the system:

a. The effectiveness and versatility of the language MODULA II,

b. The high performance in program execution achieved by LILITH because of its extremely compact compiled code and its specialized architecture,

c. The ability to show elaborate images with a high resolution display and the ability to manipulate them with powerful firmware-implemented graphic operations,

d. The efficiency for entering user commands available from the combination of the Keyboard and mouse as input devices,

e. The reliability of both software and hardware resulting from the avoidance of unnecessary complexity.

It will be shown that the LILITH computer and the MODULA II language have delivered a significant performance for a modest investment of resources.

3. Scenario of demonstration:

For groups of around 10 individuals, the demonstration will consist of a presentation of the machine as the programmer or user sees it and short statements covering respectively the innovations in the language MODULA II and the LILITH computer. The purpose of these statements will be to open discussions with the visitors and invite their inquiries for a greater depth of understanding in either the hardware or software area.

During the presentation of the machine as it is actually used, the visitor will be exposed to the graphics display capabilities and the use of the mouse for entering commands. The visitor will also see a demonstration of the text editor and the compiler in preparation for a program for execution. A graphics editor will demonstrate the preparation of a document containing a circuit diagram.

4. Literature on MODULA II and LILITH

N. Wirth, Modula: a language for modular multiprogramming, Software - Practice and Experience, 7, 3-35 (1977).

N. Wirth, MODULA-2, research report Nr. 36, Institut fuer Informatik, ETH-Zurich, 8092 Zurich, Switzerland.

J. Hoppe: A Simple Nucleus Written in MODULA-2, research report Nr. 35, Institut fuer Informatik, ETH-Zurich, 8092 Zurich, Switzerland.

5. Biographical Data:

N. Wirth, Professor, Institut fuer Informatik, ETH-Zurich

R. Ohran, Asst. Professor, Electrical Engineering Dept., Brigham Young University, on leave to Institut fuer Informatik, ETH-Zurich.

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . TEXT INPUT
. . CODE INPUT
. . . MODULA II
. . . MODULA
FUNCTION
. TRANSFORMATION
. . TRANSLATION
. . EDITING
. . OPTIMIZATION
OUTPUT
. USER OUTPUT
. . GRAPHICS
. . LISTINGS
. MACHINE OUTPUT
. . OBJECT CODE OUTPUT

IMPLEMENTATION LANGUAGE: MODULA II

COMPUTER (OTHER HARDWARE): LILITH

TOOL AVAILABLE: YES, PUBLIC DOMAIN: NO

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): LICENSED BY ETH

TOOL SUPPORTED: NO

CONTACT: RICHARD OHRAN, BRIGHAM YOUNG UNIVERSITY, EE DEPT, 459 CLYDE BUILDING, PROVO, UTAH, 84602, USA, 801-378-4015

A Demonstration of the Development of Interactive Software with Integral Help

Robert S Fenchel*
University of California, Los Angeles
March 1981

There are many complex factors which determine whether a user interface is good. One set of factors is concerned with properties that make it easy to issue desired commands after a user understands the information system. Another set of factors is concerned with interaction when a user does not understand enough about the functionality of the information system to know what to do next. In the latter case, it is obviously not possible to predict every condition and provide an adequate response. Instead, it is essential that systems have a strong *self-describing* property and that they respond to a simple user request for help. Such systems must be able to teach; i.e., to help a user to learn about their behavior.

It is the author's contention that the self-describing property should not be created as an afterthought - it can best be built if a help system is integral; i.e., designed into the system. Integral help has been incorporated into SARA (*S*ystem *A*rchitect's *A*pprentice) tools. SARA is a unique software resource supporting interactive design of information systems. The user interface to such a system is particularly important because if not done well it detracts from high-value design objectives of a user.

This demonstration presents a software engineering tool to aid in the development of interactive programs and in the process to provide integral assistance information. This tool produces programs intended to execute in a run-time environment which provides consistent access to a help function and other utilities.

Using these techniques, interactive programs can be developed and debugged. The end result runs in a sophisticated environment providing powerful user interface and user assistance capabilities. The inputs to the tool consist of the following:

1. A context free grammar specifying the interaction language of the program under development.
2. Error recovery specifications.
3. References to semantic routines to implement the functionality of the program.
4. Semantic descriptions of language constructs.

The tool processes these inputs and produces the following:

1. A parser to recognize the interaction language.
2. A data base of user assistance information.
3. The source code (with formatting commands) for a user's manual.
4. The framework for developing semantic routines.

*This work was supported by the Department of Energy, Contract No. DE-AF03-765F0034 P.A., No. DE-AT-036, ER70214, Mod. A006

Author's current address: Xerox Corporation, 701 S. Aviation Blvd., El Segundo, CA 90245

The standard run-time environment provides a consistent user interface to all programs developed using these techniques. The environment provides consistent access to the following classes of commands:

1. Input/Output stream manipulation commands.
2. Information and assistance commands.
3. Terminal characteristic specification/modification commands.
4. Inter-user communication commands.
5. Commands which afford access to operating system capabilities.
6. Commands which afford access to (specialized) storage system facilities.

Each of the commands in these classes is available uniformly throughout the use of any programs developed in this environment. The environment contains additional user interface components to ease the burden of the user in attempting to use and to learn the use of the programs available. These components include an interactive tutor to present lessons to a user and to promote controlled experimentation, as well as comment and news facilities to allow users to influence the design of programs and to be notified of any changes.

The tools and environment are available on the MIT-Multics computer system and are accessible via ARPANET, TELENET and TYMENET. Anyone interested in these facilities is encouraged to open an MIT-Multics account and access the demonstrated tools.

References

Fenchel, R. *Integral Help for Interactive Systems*, Computer Science Department, School of Engineering and Applied Science, University of California, Los Angeles, UCLA-ENG-8051, September 1980.

Fenchel,R. and G. Estrin. "Self-Describing Systems Using Integral Help", Internal Memorandum #201, University of California, Los Angeles, April 1980 (submitted to IEEE Transactions on Systems, Man and Cybernetics).

Fenchel,R. "An Integral Help System and User Interface Demonstration on UCLA's SARA (System Architect's Apprentice) System", *The UCLA Computer Science Department Quarterly*, Vol. 7, No. 3, pp. 105-124, July 1979.

Station, Day and Time

Station 3, Tuesday March 10, from 11:00 am until 3:00 pm.

Demonstrator

Robert S Fenchel

Dr. Fenchel is currently a member of the User Interface Engineering group of Xerox Corporation in El Segundo. He received Ph. D. and M.S. degrees in computer science from UCLA (1975,1980) and a B.A. degree in mathematics from the University of Rochester (1973). His interests focus on improving the interface between people and computers. His current research areas include techniques for developing interactive computer systems which provide accurate and consistent assistance for their users. Dr. Fenchel was a member of the SARA research group at UCLA and is responsible for the user interface of the SARA system.

The following is a transcript of an actual session with the tools to aid in developing interactive programs with integral help. The tools reside at MIT-Multics and are accessible via the ARPANET, TELENET or TYMENET networks. This transcript shows a user logging in to the MIT-Multics machine, invoking the grammar processing program to process the specification of the SARA News system. Once the specification has been processed successfully, the user executes the news system using other components of the development environment. At this point the syntax of the language for interactive with the news system is tested, as well as the syntactic and semantic help, and syntactic error messages and error recovery. In order to distinguish between user input and system response, all user input appears in bold faced type unless otherwise noted.

```
Multics 34.30: MIT, Cambridge, Mass.
Load = 28.0 out of 100.0 units: users = 28, 11/25/80  2342.7 est Tue
login Fenchel
Password:YourPassword
You are protected from preemption.
Fenchel SARA logged in 11/25/80  2342.8 est Tue from TELNET ASCII
terminal "NET".
Last login 11/25/80  2340.2 est Tue from TELNET ASCII terminal "NET".
line kill ~
You have no mail.
r time 23:43 cost $0.01 $0.01
```

The user invokes the grammar processor. This tool accepts a specification of the interaction language for the interactive program under design. The specification includes error recovery information and a specification of the meaning of each language construct.

```
ec grammar
SLR(1) Grammar Preprocessor  December 19, 1979
You will be prompted for the following:
   directory name -> directory to store grammar tables
   tool name      -> the name of the tool being designed
directory name>news
Working library now >user-dir-dir>SARA>Fenchel>news
tool name>news
>&input news.grammar -echo
```

The user requests that the contents of the file news.grammar be used as input to the grammar program. In this way the user can prepare the input using an editor and then feed it to the grammar program.

```
Input from source >user-dir-dir>SARA>Fenchel>news>news.grammar

 started
/* This is the grammar for the SARA news system */
TITLE "News System"
VERSION "SLR(1) version 1.3, includes descriptions"
AUTHOR "R. Fenchel"

/*  set the default for the error recovery */
DEFAULT R News-Command S ";" ;
```

The specification of the grammar (interaction language) is a BNF-like notation for a context free language. Notice that several additonal components have been added. The *D !* precedes a semantic description. This description is used to generate interactive assistance as well as user manuals. *E !* and *N !* indicate example and note sections respectively. Notice that all three of these sections may include formatting instructions (e.g. *.li 1*). The information surrounded by *%* characters specifies the semantic routine (if any) to be invoked when its associated production is reduced by the generated parser. Syntactic error recovery and other production or nonterminal specific information may also be included. For example, the notation *%I 2%* indicates that semantic routine number 2 is to be associated with the current production.

```
/* now specify the grammar */
News-System : News-Command-list
            D ! The news system provides SARA users the
              ability to learn of any changes to the SARA
              system, or any other SARA related information.
              The user may list the names of "news" segments
              and view the contents of these segments. !      ;
News-Command-list : News-Command-list  News-Command
                                       Command-Terminator
                  | News-Command Command-Terminator
      D !News commands are terminated by a new line character.
         multiple news commands may be placed on a line by
         separating them by a semicolon.!;
News-Command : Display-Menu-Command
             | Print-News-Command
             | End-Command
             |
      D !A News command allows the user to do one of the
         following:
         .li 1
         .le
         Display a menu of available news items
         .le
         Select and print the contents of a news item
         .le
         End the news tool and return to the SARA selector
         .el  ! ;
Display-Menu-Command : @menu Date-option %I 1 %
            D ! The display menu command provides the
              ability to list the names of news items.
              The user may indicate an optional date
              and thus request that only names of news items
              modified
              on or after the given date be displayed.
              !
            E !@menu 4/5/79 !
              ;
```

```
Print-News-Command : @print News-Item    %I 2%
      D ! The print news command allows the user to view the
          contents of the indicated news item.!
        N !News-Item must be one of the items
          available in the news system

            (see Display-Menu-Command).!
                 E !@print help
                    .br
                    @print plip-installed !;
End-Command : @end                          %I 3%
            D ! The end command ends the SARA News tool. !
            E ! @end!;

Date-option : Date          %I 8%
            |           %I 7%
            D ! A date option is an optional date.  If the
              date is indicated, then news items on or after
              that date are selected.  If no date is given,
              then ALL news items are selected.  !   ;
Date : Month Delimiter Day Year-option
      D !A date is a calendar date where the year specified is
         optional.!
      E !3/16/79
         .br
         9/21 !;
Year-option : Delimiter Year
            |           %I 10%
      D !A year option is an optional specification of a year.
         If a year is not given, then the current year is
         assumed.! ;
Month : integer %I 11%
       D !A month is a numerical (integer) specification of
          a month of the year!
        N !The month integer must be between 1 and 12 inclusive
Day   : integer %I 13%
      D !A day is a numerical (integer) specification of
         a day of a month!
      N !The day integer must be valid for whatever month
         is being used.! ;
Year   : integer %I 9%
      D !A year is a numerical (integer) specification of a
         calendar year.  The year must be specifed by the
         last 2 digits of the calendar year only.!
      E !78
         .br
         82 !;
Delimiter : /
          | .
          | -
      D !A delimiter is a symbol used to separate the various
         components of a date.! ;
News-Item : id %I 4%
          | qualified-id %I 4%
          D ! A news item is the name of any available news
            item.  The list of available news items is
            generated by the "@menu" command
            (see Display-Menu-Command). !;
Command-Terminator : ";"   % X %
                   | new-line   % X %
      D !A command terminator is used to indicate the end
         of a news system command.  The ";" may be used to
         place multiple commands on one input line.! ;

/* Specification of synonyms for terminals in language */
@menu = @mn @m ;
@print = @pr @p ;

END
```

The input to the grammar program is finished and processing is in progress. Any errors or warnings will be presented to the user via the user's terminal, a detailed listing of the results of the grammar program are included in a file. If the processing completes satisfactorily, the grammar postprocessor will generate several files which will include parse tables, help information and user manual source.

```
Listing will appear in file: [pd]>news.listing

Intermediate grammar processing started.
***  the grammar is slr(1)  ***
Intermediate grammar processing completed.

Grammar Postprocessor
Working library now >user-dir-dir>SARA>Fenchel>news
news grammar is slr(1)
End of Grammar Postprocessor
Converting parse tables to SARA format.
End of SLR(1) Grammar Preprocessor
r time 00:07 cost $1.08 $1.10
```

The grammar processing has been completed successfully. Now the user tests the language which has been constructed by using the input/output system, lexer, parser etc. in the run-time environment associated with the development environment. At this point, no semantic routines have been implemented, thus valid syntax will be recognized, however no semantic processing will take place. Notice that syntactic and semantic help are available at this stage, as are error messages and error recovery. Also note that assistance may be invoked (via the *?* facility) at any point of system use. This is indicted by the *?* appearing both preceding and following commands in this example.

```
ec test.grammar >udd>SARA>Fenchel>news news
News System  November 26, 1980
>?
Expecting News←Command or Command←Terminator
News←Command starts with one of the following:
   @menu  @print  @end
Command←Terminator starts with:
   ;
   or input of type:
   new←line
>?@menu
Syntax:
Display←Menu←Command ->  @menu Date←option
>?@menu +manual
Description:
The display menu command provides the ability to list the
names of news items.  The user may indicate an optional date
and thus request that  only names of news items modified on
or after the given date be displayed.
Syntax:
Display←Menu←Command ->  @menu Date←option
Example:
@menu 4/5/79
>@menu ?
Expecting Date←option or Command←Terminator
Date←option starts with input of type:
   integer
Command←Terminator starts with:

   ;
   or input of type:
   new←line
@menu >12?
Expecting Delimiter
Delimiter starts with one of the following:
   /  ,  -
@menu 12>/?
Expecting Day
Day starts with input of type:
   integer
@menu 12/>21
```

The following are examples of generated syntactic error information. Notice the terse error message and the way in which the user can easily request additional information for components of the error message (e.g. *News←Command*).

```
>ERROR
ERROR
↑
News←Command or Command←Terminator was expected
>@bad command
@bad command
↑
News←Command or Command←Terminator was expected
>?news←command +manual
Description:
A News command allows the user to do one of the following:

1.   Display a menu of available news items

2.   Select and print the contents of a news item

3.   End the news tool and return to the SARA selector

Syntax:
News←Command-> Display←Menu←Command
News←Command-> Print←News←Command
News←Command-> End←Command
News←Command->
>?end←command
Syntax:
End←Command-> @end
>@end
r time 00:12 cost $0.17 $1.27
```

The testnog session has been terminated, at this point the user may wish to modify the language specification. In this example, the user decides to quit and thus terminates the Multics session.

```
logout
```

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . VHLL INPUT
. . . SL1
. . . GMB
. . . BNF
. . TEXT INPUT
FUNCTION
. TRANSFORMATION
. . TRANSLATION
. . FORMATTING
. . RESTRUCTURING
. STATIC ANALYSIS
. . DATA FLOW ANALYSIS
. . STRUCTURE CHECKING
. . CROSS REFERENCE
. . CONSISTENCY CHECKING
. . COMPLETENESS CHECKING
. . SCANNING
. DYNAMIC ANALYSIS
. . SIMULATION
OUTPUT
. USER OUTPUT
. . GRAPHICS
. . LISTINGS
. . USER-ORIENTED TEXT
. . . DOCUMENTATION
. . . ON-LINE ASSISTANCE
. MACHINE OUTPUT
. . SOURCE CODE OUTPUT
. . DATA OUTPUT
. . PROMPTS

IMPLEMENTATION LANGUAGE: PL/1, TOOL PORTABLE: NO, TOOL SIZE: 25000 LINES OF PL/1 SOURCE

COMPUTER (OTHER HARDWARE): HONEYWELL, OS (OTHER SOFTWARE): MULTICS

TOOL AVAILABLE: YES, PUBLIC DOMAIN: YES, TOOL SUPPORTED: NO

CONTACT: G. ESTRIN, UNIVERSITY OF CALIFORNIA, COMPUTER SCIENCE DEP, BOELTER HALL 3732, LOS ANGELES, CA, 90024, USA, 213-825-8878

THE SOFTWARE TOOLS VIRTUAL OPERATING SYSTEM PROJECT

Debbie Scherrer, Dave Martin, Chris Petersen

1. Virtual Operating Systems

One of the many problems which programmers (and end users) encounter in their everyday use of computers is the lack of common utilities as they move from system to system. In addition, moving code from machine to machine is costly and error-prone. These problems can be reduced through the use of a virtual operating system that disentangles computing environments from their underlying operating systems. The Software Tools project is an experiment to achieve inter-system uniformity at all levels of user interface.

A real operating system presents three principal interfaces to its users: the virtual machine or operating system primitives accessible through programming languages; the utilities programs such as editors, compilers, linkers; and the command language or means by which users access system resources from a terminal. The idea of a virtual operating system is to provide standard versions of these interfaces, based on organizational requirements. Possible applications include data management environments, office information environments, real-time process control environments, and program development environments, to name a few.

2. The Software Tools

The Software Tools to be demonstrated represent a program development environment as one realization of a virtual operating system. The environment consists of resources which assist programmers in the development and maintenance of computer programs. The system is modelled after the UNIX* operating system, a system considered by many to provide a superior collection of program development aids. Many of the utilities originated in the book "Software Tools" by Brian Kernighan and P. J. Plauger.

To be useful, the Software Tools environment was designed to be portable and to co-exist with the local operating system. More than 300 different sites, running 50 operating systems, have implemented these tools to varying degrees of sophistication.

A users group has also been organized. Current activities include the establishment of a centralized tape distribution facility, distribution of a newsletter, organization of active special interest groups on various topics, and sponsorship of biannual meetings.

* UNIX is a registered trademark of Bell Labs

3. The Demonstration

The demonstration will begin with a description of the virtual operating system approach, its advantages and disadvantages. With this as background, we will then concentrate on a description of the utilities, both those provided with the basic package and experimental extensions. Emphasis will be on the text manipulation functions and on the powerful command line interpreter, which assists the user in combining small tools to perform larger, more complex tasks. Finally, we will present several problems to the audience either for discussion or direct solution on the machine.

To emphasize the portability of the package, we hope to present the tools running on a large minicomputer and on a small Z80-based micro.

4. References

The following documents will be available for reference at the demonstration:

Akin, T. Allen, P. Flinn, and D. Forsyth, Jr., 'Software Tools Subsystem Reference Manual', Technical Report, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA, April 1978.

Hall, Dennis E., Deborah K. Scherrer, Joseph S. Sventek, 'The Software Tools Programmers Manual', Internal Rep. LBID 097, LBL, University of California, Berkeley, CA, 1978.

Hall, Dennis, Deborah K. Scherrer, Joseph S. Sventek, 'A Virtual Operating System,' Communications of the ACM, Vol. 23 (Sept 80), pp. 495-502.

Hanson, David R., 'The Portable Directory System PDS', Technical Report TR 80-4, Department of Computer Science, The University of Arizona, Tucson, AZ, March 1980.

Hanson, David R., 'The Portable I/O System PIOS', Technical Report TR 80-6, Department of Computer Science, The University of Arizona, Tucson, AZ, April 1980.

Hanson, David R., 'Software Tools Programmer's Manual', Technical Report TR 79-15, Department of Computer Science, The University of Arizona, Tucson, AZ, August 1979.

Kernighan, Brian W., 'Ratfor - A Preprocessor for a Rational Fortran', Software - Practice and Experience, Vol. 5, 4 (Oct-Dec 75), pp. 395-406.

Kernighan, Brian W, and P. J. Plauger, Software Tools, Addison-Wesley Publishing Company, Reading, MA, 1976.

Scherrer, Deborah, 'COOKBOOK - Instructions for Implementing the Software Tools Package', Technical Report LBID 098, Department of Computer Science and Applied Mathematics, Lawrence Berkeley Laboratory, University of California, Berkeley, CA.

"Machines and Operating Systems Running the Software Tools"; a list of names and addresses of people to contact who have or who are interested in implementing the software tools on various systems.

"Software Tools Communications" - the newsletter distributed by the Software Tools Users Group.

5. Location and Time

Station 4, Tuesday, March 10 from 9:00 AM to 5:00 PM.

6. Personnel

The demonstration will be prepared and led by the following people:

Debbie Scherrer
Lawrence Berkeley Laboratory
Computer Scientist and member of the Advanced Systems research group at LBL. Along with her colleagues Dennis Hall and Joe Sventek, one of the founders of the virtual operating system approach. Organizer of the Software Tools Users Group.

Dave Martin
Hughes Aircraft Company
Staff Engineer at Hughes. Dave is a prolific toolsmith whose primary interest is in extensions to the basic software tools package. His experimental command line interpreter will be available for use in the demonstration.

Chris Petersen
ORINCON Corporation
Principal Engineer and manager of computer services at ORINCON; primarily concerned with the development of software systems for various Navy applications. Implemented the software tools on the TENEX system.

THE SOFTWARE TOOLS UTILITIES

MAINTAINING FILES

Looking at/copying files

cat ..... concatenate and print text files
crt .....copy files to terminal
pl ..... print specified lines/pages in a file
pr ..... print file
show ..... show all characters in a file
tail ..... print last lines of a file
tee ..... copy input to output and named files

Organizing Files

cd ..... change directory
ls ..... list current directory
mv ..... move (rename) a file
pwd ..... print working directory name
rm ..... remove (delete) files

Grouping Files

ar ..... archive file maintainer
includ ..... file inclusion preprocessor
lam ..... laminate files
split ..... split file into pieces

Monitoring Files

cmp ..... comparing files
comm ..... print lines common to two files
diff ..... isolate differences between files
ll ..... print line lengths
uniq ..... strip adjacent repeated lines from a file
wc ..... count lines, words, and characters in files

MANIPULATING TEXT

Altering Text

ch ..... change text patterns
ed ..... editor
sedit ..... stream editor
tr ..... character transliteration

Transforming Text

cpress ..... compress input files
crypt ..... encrypt and decrypt standard input
detab ..... convert tabs to spaces
entab ..... convert spaces to tabs and spaces
expand ..... uncompress input files
os ..... convert backspaces into multiple lines

Arranging Text

field ..... manipulate fields of data
kwic ..... prepare lines for keyword-in-context index
mcol ..... multicolumn formatting
rev ..... reverse lines
roff ..... text formatter
sort ..... sort and/or merge text files
tsort ..... topologically sort symbols
unrot ..... unrotate lines prepared by kwic

Locating Text

fb ..... search blocks of lines for text patterns
find ..... search a file for text patterns
spell ..... locate spelling errors
xref ..... make a cross reference of symbols

Language Translation

macro ..... general-purpose macro processor
ratfor ..... Ratfor preprocessor
rc ..... Ratfor compile, link, and load
fc ..... Fortran compile, link, and load
ld ..... Link and load

MONITORING THE ENVIRONMENT

Process Control

sh ..... command line interpreter
pstat ..... show status of process
kill ..... kill process
suspnd ..... suspend process
resume ..... resume suspended process

User Support

dc ..... desk calculator
echo ..... echo command line arguments
man ..... show users manual page
help ..... user assistance

Information Retrieval

date ..... print the date and time
who ..... list who is on the system
users ..... list valid (mail) users

Inter-user/inter-machine communication

mail ..... send/receive mail
msg ..... fancy mailer

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . CODE INPUT
. . . RATFOR
. . TEXT INPUT
FUNCTION
. TRANSFORMATION
. . EDITING
. . TRANSLATION
. . FORMATTING
. STATIC ANALYSIS
. . MANAGEMENT
. . . FILES MANAGEMENT
. . CROSS REFERENCE
. . COMPARISON
OUTPUT
. USER OUTPUT
. . LISTINGS
. . DIAGNOSTICS
. . USER-ORIENTED TEXT
. MACHINE OUTPUT
. . SOURCE CODE OUTPUT
. . . FORTRAN

IMPLEMENTATION LANGUAGE: FORTRAN, PASCAL, RATFOR

TOOL PORTABLE: YES, TOOL SIZE: 50000 LINES

TOOL AVAILABLE: YES, PUBLIC DOMAIN: YES

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): SOME SPECIFIC SYSTEMS INTERFACES ARE BEING MARKETED

TOOL SUPPORTED: YES, TOOL SUPPORT: SOFTWARE TOOLS USERS GROUP

CONTACT: MICHAEL BOURKE, SRI INTERNATIONAL, 333 RAVENSWOOD AVE., PN311, MENLO PARK, CA, 94025, USA, 415-326-6200

STRUCTURE CHART GRAPHICS SYSTEM

A Software Engineering Tool Demonstration

Debra L. Resendez and James W. Winchester

1. Introduction

This proposal is in response to the call for software engineering tool developers to demonstrate their accomplishments at the 5th International Conference on Software Engineering in San Diego. March 9-12, 1981.

We will be demonstrating the SCG system developed by Hughes Aircraft Company's Software Engineering Division as an Internal Research & Development project.

2. Summary of SCG

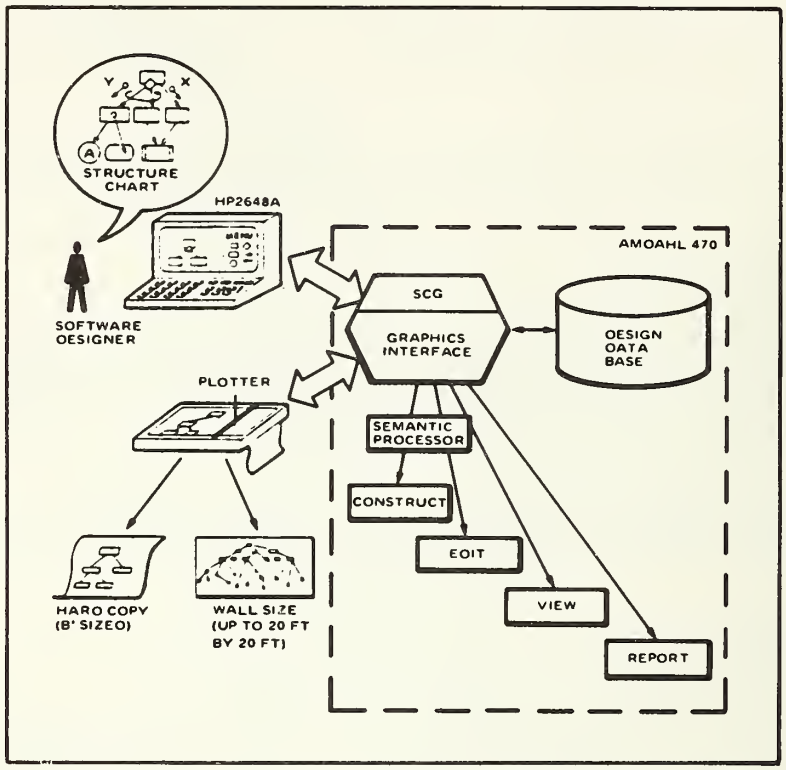
The Structure Chart Graphics subsystem (SCG) was developed to solve the following fundamental software design problem:

> Although effective design guidelines have been established (such as with Yourdon, Meyers, & Constantine) which reduce the overall life cycle cost of software, consistent and thorough application of these guidelines across all development efforts is hampered by a lengthy technology transfer time and the rare availability of expert designers.

The recommended three-step approach to easing this problem is, first, to develop a user-friendly graphics interface which provides automatic documentation. Second, once having captured a design in a machine readable form, develop automatic design quality analyzers to enforce adherence to design standards. Third, gain acceptance and widespread use. The SCG depicted below is a first generation feasibility demonstration of this approach.

The user interacts with SCG through a graphics terminal using design commands such as "place a module", "connect two modules", and using graphics commands such as "jump to module x", "reposition subtree to (x,y)". Having captured the design in a data base, the user can request either page-sized CALCOMP copies or wall charts of the entire design. SCG is a feasibility prototype and has led to the

specification of a computer aided design tool called Automated Interactive Design and Evaluation System (AIDES) described in [Willis79].



Overview of the SCG. Interactive graphics and automatic documentation are key features.

3. A Scenario of an SCG Demonstration

A sample structure chart will be developed using the SCG commands. The meaning of the structure chart will be described while developing it at the terminal. Samples of structure chart plotted output will be available for hand-out. ISCE attendees will be permitted to enter commands to add to or modify the sample structure chart. A users manual will be available.

4. SCG Literature

SCG related literature includes:

SCG Users Manual

[WILLIS79] Willis, R. R., E. P. Jensen. "Computer Aided Software Systems", Proc. 4th International Conference on Software Engineering, September 17-19, 1979.

[YIN79] Yin, B., J. W. Winchester. "Software Design Quality Metrics System," The Second International Conference on Mathematical Modeling, St. Louis, Missouri, July 1979.

5. Station, Day, and Time

Station 5, Tuesday, March 10, from, 9:00 a.m until 7:00 p.m.

6. The Demonstrators

Debra L. Resendez

Debra Resendez is a member of the Technical Staff for Software IR&D at Hughes Aircraft Company. She is currently involved in the specification of the requirements for an Integrated Software Development Facility (ISDF) using a formal requirements definition language. Her background is in requirements definition methodologies and data base interfaces for interactive software systems. Previous activities include extensive enhancements to the Structure Chart Graphics (SCG) System; an interactive tool for creating and maintaining structure charts. Ms. Resendez holds a B.S. in Computer Science from the University of North Dakota and is currently pursuing an M.S. in Electrical Engineering -Computers at the University of Southern California.

James W. Winchester

James Winchester is currently the Head of the Research & Analysis Section in the Software Engineering Division at Hughes Aircraft Company. Dr. Winchester is responsible for ongoing research and development in the areas of software and system specification, design and development. Dr. Winchester was previously Head of the Research Analysis Group leading software IR&D projects. His particular research emphasis is in requirements specification languages and their relationship to the system development life cycle. Prior to his employment with Hughes, Dr. Winchester was an officer in the U. S. Army, where he directed the development of an automated maintenance management system. He holds a B.S. degree in Engineering Physics from Cornell University, as well as an M.A. (Education) and Ph.D. (Computer Science) from U.C.L.A.

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . VHLL INPUT
. . . DESIGN SPECIFICATION
FUNCTION
. TRANSFORMATION
. . TRANSLATION
. . RESTRUCTURING
OUTPUT
. USER OUTPUT
. . GRAPHICS
. . . STRUCTURE CHARTS

IMPLEMENTATION LANGUAGE: FORTRAN

TOOL PORTABLE: PARTIAL

COMPUTER (OTHER HARDWARE): AMDAHL 470 (HP2647A OR HP2648A GRAPHICS TERMINAL, CALCOMP PLOTTER), VAX 11/780 (HP2647A OR HP2648A GRAPHICS TERMINAL, CALCOMP PLOTTER)

OS (OTHER SOFTWARE): MVS (PLOT-10, ADBMS)

TOOL AVAILABLE: NO, PUBLIC DOMAIN: NO

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): HUGHES PROTRIETARY

TOOL SUPPORTED: NO

CONTACT: JAMES W. WINCHESTER, HUGHES AIRCRAFT COMPANY, POST OFFICE BOX 3310, FULLERTON, CA, 92634, USA, 714-732-3232

DESIGN QUALITY METRICS SYSTEM

A Software Engineering Tool Demonstration

Debra L. Resendez and James W. Winchester

1. Introduction

This proposal is in response to the call for software engineering tool developers to demonstrate their accomplishments at the 5th International Conference on Software Engineering in San Diego, March 9-12, 1981.

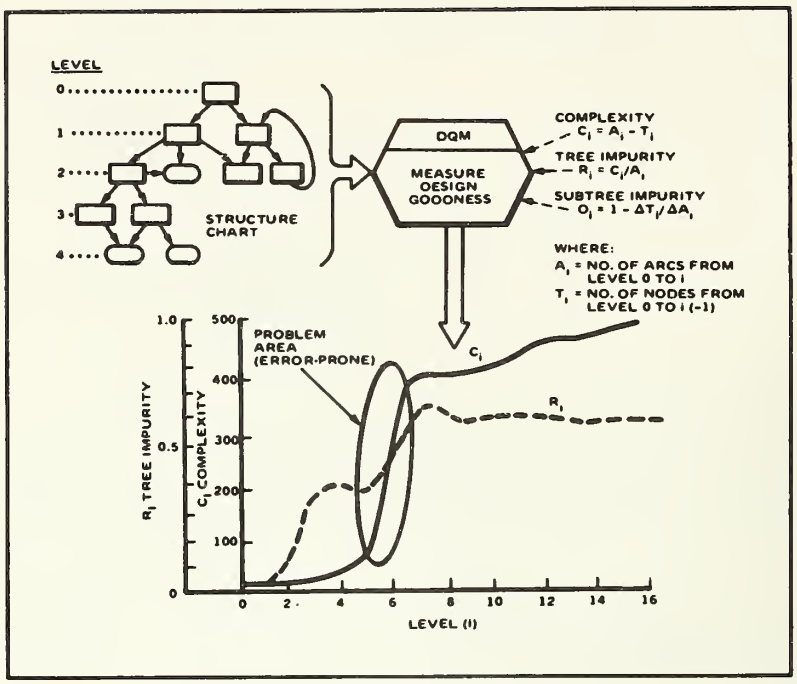
We will be demonstrating the DQM system developed by Hughes Aircraft Company's Software Engineering Division as an Internal Research & Development project.

2. Summary of DQM

Although methodologies with qualitative guidelines abound, few, if any, are supported with quantitative, measurable means of adherence. Indeed, Hughes' Structured Design methodology for software decomposition contains guidelines such as "maximize modularity" and "minimize coupling" - both of which have no apparent quantifiable measure of compliance.

The Design Quality Metrics System (DQM) is an initial step to solve this problem. As shown below and reported in [Yin78], [Yin79] algorithms have been developed which, when applied to the structure chart of software modules, produce a quantification of the design using a plot of complexity as a function of tree depth. The results of this technique have been validated on two major software development efforts and have been shown to correlate closely with the number of errors encountered in software testing. In other words, DQM quantifies qualitative guidelines and, therefore, provides a predictive tool which can reduce the number of errors and the cost of software.

DQM obtains design information from the Structure Chart Graphics data base. The user can interactively request plots of complexity for any subtree, fan-in and fan-out information, and lists of module names contained in any subtree of the hierachy.



DQM (Design Quality Metric) System. A quantitative measure for SW design "goodness".

3. A Scenario of a DQM Demonstration

A sample structured design will reside in the SCG data base. DQM will be invoked to measure the complexity of this sample design. Plots of complexity, module "call" information, and fan-in/fan-out data will be displayed. The sample design will then be modified via SCG commands and the DQM re-invoked to portray the use of DQM in achieving designs which are less complex. Handouts will be available. ISCE attendees will be permitted to interactively modify a design and measure its quality.

4. DQM Literature

DQM related literature includes:

DQM Users Manual

[YIN78] Yin, B., J. W. Winchester. "The Establishment and Use of Quality Measures to Evaluate the Quality of Software Designs," Proc. ACM Software Quality Workshop, San Diego, Ca. November 1978.

[YIN79] Yin, B., J. W. Winchester. "Software Design Quality Metrics System," The Second International Conference on Mathematical Modeling, St. Louis, Missouri, July 1979.

5. Station, Day, and Time

Station 5, Tuesday, March 10, from 9:00 a.m. until 7:00 p.m.

6. The Demonstrators

Debra L. Resendez

Debra Resendez is a member of the Technical Staff for Software IR&D at Hughes Aircraft Company. She is currently involved in the specification of the requirements for an Integrated Software Development Facility (ISDF) using a formal requirements definition language. Her background is in requirements definition methodologies and data base interfaces for interactive software systems. Previous activities include extensive enhancements to the Structure Chart Graphics (SCG) System; an interactive tool for creating and maintaining structure charts. Ms. Resendez holds a B.S. in Computer Science from the University of North Dakota and is currently pursuing an M.S. in Electrical Engineering -Computers at the University of Southern California.

James W. Winchester

James Winchester is currently the Head of the Research & Analysis Section in the Software Engineering Division at Hughes Aircraft Company. Dr. Winchester is responsible for ongoing research and development in the areas of software and system specification, design and development. Dr. Winchester was previously Head of the Research Analysis Group leading software IR&D projects. His particular research emphasis is in requirements specification languages and their relationship to the system development life cycle. Prior to his employment with Hughes Dr. Winchester was an officer in the U. S. Army where he directed the development of an automated maintenance management system. He holds a B.S. degree in Engineering Physics from Cornell University, as well as an M.A. (Education) and Ph.D. (Computer Science) from U.C.L.A.

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . VHLL INPUT
. . . DESIGN SPECIFICATION
FUNCTION
. TRANSFORMATION
. . TRANSLATION
. STATIC ANALYSIS
. . COMPLEXITY MEASUREMENT
OUTPUT
. USER OUTPUT
. . GRAPHICS
. . . DESIGN CHARTS
. . TABLES

IMPLEMENTATION LANGUAGE: FORTRAN

TOOL PORTABLE: PARTIAL

COMPUTER (OTHER HARDWARE): AMDAHL 470 (HP2647A OR HP2648A GRAPHIC TERMINAL)

OS (OTHER SOFTWARE): MVS (ADBMS AND PLOT-10)

TOOL AVAILABLE: NO, PUBLIC DOMAIN: NO

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): HUGHES PROPRIETARY

TOOL SUPPORTED: NO

CONTACT: JAMES W. WINCHESTER, HUGHES AIRCRAFT COMPANY, POST OFFICE BOX 3310, FULLERTON, CA, 92634, USA, 714-732-3232

AUTOMATED INTERACTIVE SIMULATION MODELLING SYSTEM

A Software Engineering Tool Demonstration

William P. Austell Jr. and Ronald R. Willis

1. Introduction

This proposal is in response to the call for software engineering tool developers to demonstrate their accomplishments at the 5th International Conference on Software Engineering in San Diego, March 9-12, 1981.

We will be demonstrating the AISIM system, currently under development by Hughes Aircraft Company's Software Engineering Division for the Air Force's Electronics Systems Division at Hanscom Air Force Base, Massachusetts.

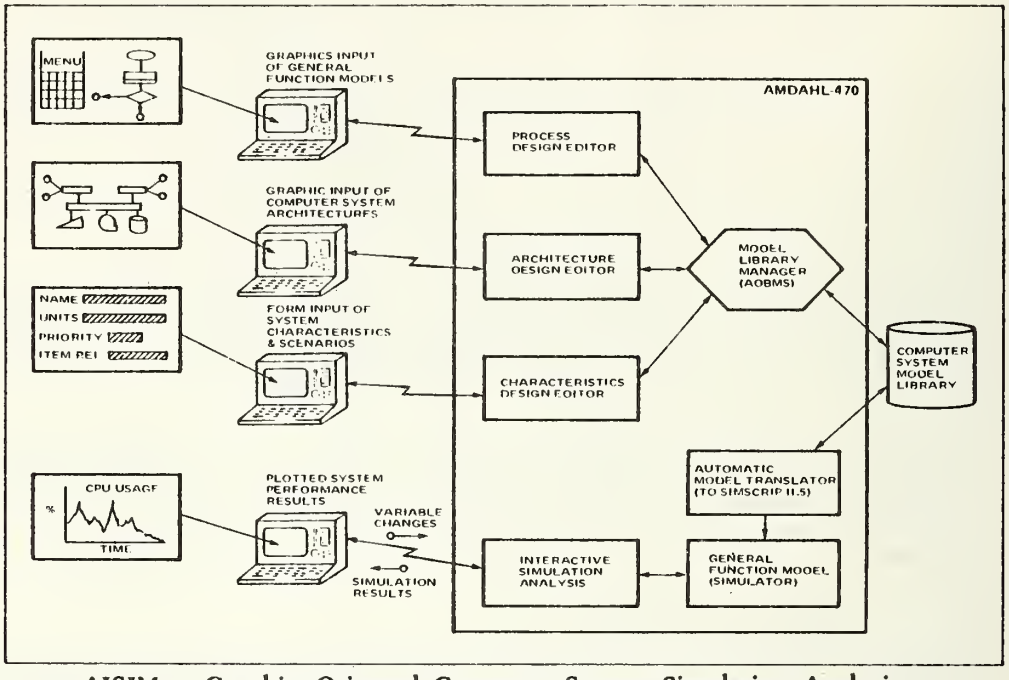
2. Summary of AISIM

The Automated Interactive Simulation Modelling System (AISIM) is an outgrowth of two previous modelling systems: The Design Analysis System (DAS), which provided interactive graphics support for modelling general functions in a procedure-oriented system, and the Distributed Data Processing Model (DDPM), which consisted of a library of standard, table-driven computer system models. AISIM combines the features of both to provide an interactive, graphics simulation system for distributed system analysis.

As shown in the figure below AISIM provides three types of interfaces for modelling computer systems. Flowchart-oriented processes representing software, information flow, or man/machine procedures are entered via an interactive graphics language consisting of execution control, resource allocation, and timing primitives. Architectures of interconnected processors, channels, disks, tapes, and other devices are graphically entered via an interactive architecture design editor. Device characteristic specifications, scenario definitions, and simulation variables are entered via an interactive form-oriented editor.

After a model has been developed, it can be automatically translated into an executable SIMSCRIPT II.5 simulation model. Then the user can simulate the computer system interactively while stopping at specified breakpoints to

change critical system variables and observe cause/effect relationships. Outputs are interactive user-defined plots and complete statistical summaries in listing form.



AISIM – Graphics-Oriented Computer System Simulation Analysis

3. A Scenario of an AISIM Demonstration

A sample distributed data processing model will reside in the AISIM data base. AISIM commands will be exercised to view the various forms supporting the user interface. The analysis portion of AISIM will be invoked to demonstrate automatic model creation and simulation. During simulation, model variables will be changed and results plotted to show cause/effect relationships and to demonstrate interactive simulation analysis. A user's manual will be available. Copies of selected user input forms and output reports will be available.

4. Literature

AISIM related documentation includes:

[AUSTELL80] Austell, W. "AISIM Systems User's Manual." CDRL #104 of Contract F19628-79-C-0153, Electronic Systems Division (ESD), TOIT, Hanscom AFB, MA.

[AUSTELL80] Austell, W. "AISIM Final Report," CDRL #114 of Contract F19628-79-C-0153, Electronic Systems Division (ESD), TOIT, Hanscom AFB, MA.

[WILLIS78] Willis, R. R. "DAS-An Automated System to Support Design Analysis," Proc. 3rd International Conference on Software Engineering, May 1978.

5. Station, Day, and Time

Station 5, Tuesday, March 10, from 9:00 a.m. until 7:00 p.m.

6. The Demonstraters

William P. Austell, Jr.

William Austell is a project manager in the Software Engineering Division of Hughes Aircraft Company. Currently he is managing the development of two automated simulation systems; AISIM which is used to model and simulate distributed data processing systems and IDSS which is used to analyze manufacturing systems. His background includes design and analysis of various software and hardware systems. Prior to his employment with Hughes Aircraft Company, Mr. Austell was an officer in the United States Air Force where he managed the development of command, control and communication sytems. He holds a B.S. degree in Electrical Engineering and an M.S. in Electrical and Computer Engineering, both from Clemson University.

Ronald R. Willis

Ron Willis is currently the Technical Director for the Software Engineering and Technology Department in the Software Engineering Division of Hughes Aircraft Company. Mr. Willis' recent management experience has entailed simulation analysis, IR&D studies, marketing and sales, and training. He has led over 60 simulation analysis studies, ranging the spectrum of computer system applications. Earlier, Mr. Willis led IR&D studies in distributed processing computer systems, requirements and analysis using simulation, structured design, and system engineering language technology transfer. He has gained modeling training experience as an ATEP instructor for in-house courses. Mr. Willis' background includes extensive work with over 20 computer architectures, 15 operating systems, and 25 programming languages and is the author of eight papers on state-of-the-art simulation and system analysis methodologies. Mr. Willis holds a B.S. degree in Math from C.S.U. Long Beach and an M.S. degree in Computer Science from U.S.C.

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . VHLL INPUT
. . . DESIGN SPECIFICATION
FUNCTION
. TRANSFORMATION
. . TRANSLATION
. DYNAMIC ANALYSIS
. . SIMULATION
. . . COMPUTER SYSTEM SIMULATION
OUTPUT
. USER OUTPUT
. . GRAPHICS
. . TABLES

IMPLEMENTATION LANGUAGE: FORTRAN, SIMSCRIPT II.5

TOOL PORTABLE: PARTIAL

COMPUTER (OTHER HARDWARE): AMDAHL 470/V8 (HP2647/48 GRAPHICS TERMINAL)

OS (OTHER SOFTWARE): MVS (PLOT 10, SIMSCRIPT II.5, ADBMS (U. OF MICH))

PUBLIC DOMAIN: YES

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): AVAILABLE JAN 1982 THROUGH USAF/ESD/TOIT, HANSCOM AFB, MASS.

TOOL SUPPORTED: NO

CONTACT: BILL AUSTELL, HUGHES AIRCRAFT COMPANY, PO BOX 3310, FULLERTON, CA, 92634, USA, 714-732-3232

NAME OF TOOL: COBOL Structuring Engine (THE ENGINE)

TITLE: A methodology and set of software tools for introducing Structured Programming into unstructured (spaghetti) COBOL programs.

INTRODUCTION:

There will be a detailed concept presentation on our COBOL Structuring Engine (THE ENGINE) and its corresponding STRUCTURED RETROFIT methodologies given on Wednesday, 11 March 1981 at 2-3:30 P.M. The presentation will cover the purpose of Structured Retrofit, its focus,scope, methodology and success to date. The software tools used in STRUCTURED RETROFIT including the COBOL Structuring Engine will be demonstrated on Tuesday, 10 March at stage #6 throughout the day. We recommend you attend a live demonstration of the Structured Retrofit software tools to include the COBOL Structuring Engine sometime during the day on Tuesday and then attend the concept presentation on Wednesday.

SUMMARY:

Structured Retrofit is a software tools based methodology. It introduces today'sstructured programming methodologies into existing systems in order to meet future demands for change. Its focus is on application systems maintenance.

Today maintenance consumes 70% of the typical MIS budget. The structured programming methodologies cut maintenance costs by a ratio of 3 to 1. Structured Retrofit allows an organization to introduce the structured methodologies into systems developed without them in a straightforward, fast and reliable manner.

Retrofit takes working COBOL programs--written and maintained by a variety of programmers in a variety of styles--and restructures and reformats them mechanically. It gives the code a consistent structure and format, making it more readable, understandable, and maintainable. It is the application of the structured programming constructs to existing "spaghetti" code.

SCENARIO OF DEMONSTRATION

The Structured Retrofit process is reviewed and the on/line tools are demonstrated. Out put from the Retrofit process is exhibited. Materials are of a technical nature and assume participant has some knowledge of Structured Programming and the COBOL language.

LOCATION AND TIME

DEMONSTRATIONS:	DATE:	Tuesday, 10 March 1981
	TIME:	9 A.M. - 7 P.M.
	LENGTH:	1 Hour (on the hour)
	LOCATION:	Space #6

REFERENCES

Lyons, M.J., "Structured Retrofit - 1980". Proceedings of SHARE 55., (Vol. 1), 1980, pp. 263-275.

Lyons, M.J., "Salvaging Your Software Asset", Proceedings of the National Computer Conference (NCC81), Chicago, IL., May 1981, pp. (Not Yet Published).

SOFTWARE TOOL DEMONSTRATORS: JON CRIS MILLER
PRESIDENT

MICHAEL J. LYONS
VICE PRESIDENT
Marketing Director

THE CATALYST CORPORATION
433 S. KENSINGTON AVE.
LA GRANGE, ILLINOIS 60525
(312) 352-5422

```
16.13.11        MAR  4,1981

0220      MOVE  NUDAYS TO NR-NO.
0230      MOVE  UR-RUNIND TO NR-RUNIND.
0240      WRITE  NEWR FROM NEW-REC.
0250      IF  NUDAYS = 1
0260          GO TO CLOSE-OLD-FILE.
0270 READ-WRITE.
0280      READ  OLD-FILE INTO OLD-REC
0290          AT END GO TO CLOSE-OLD-FILE.
0300      WRITE  NEWR FROM OLD-REC.
0310      GO TO READ-WRITE.
0320 CLOSE-OLD-FILE.
0330      CLOSE  OLD-FILE WITH LOCK.
0340      GO TO ABAA-READ-CARDS.
0350 NEW-DATE-CHECK.
0360      IF  OKD (INDX) > EDIT-DATE
0370          MOVE  1 TO DATE-ERR.
0380 NDC-EXIT.
0390      EXIT.
0400 ABAA-READ-CARDS.
0410      READ  CARD-FILE INTO CARD-FORMS
0420          AT  END  GO TO  AXAB-SORT-RELEASE-REC.
0430      MOVE  ZERO TO RR-SUFFIX, RR-CHAN-FLD, EFI, RET-SW.
0440      MOVE  SPACES TO RR-PREFIX.
0450 ABAB-KEY-TEST.
0460      IF  CF-NO-ALPHA NOT NUMERIC
0470          ADD  1 TO EFI
0480          MOVE  'A' TO RRP-IND (EFI).
0490      IF  NOT  CF-NO-ONE-TO-FIVE
0500          ADD  1 TO EFI
0510          MOVE  'A' TO RRP-IND (EFI).
0520      MOVE  CF-NO-ALPHA  TO RR-SUFFIX.
0530      IF  NOT CF-TRAN-VALUE
0540          ADD  1 TO EFI
0550          MOVE  'B' TO RRP-IND (EFI).
0560      IF  CF-KEY-TYPE NOT NUMERIC
0570          ADD  1 TO EFI
0580          MOVE  'C' TO RRP-IND (EFI).
0590      IF  CF-KEY-UNIT NOT NUMERIC
0600           ADD 1 TO EFI
0610          MOVE  'C' TO RRP-IND (EFI).
0620      IF  RR-PREFIX NOT = SPACES
0630          MOVE  ' BAD KEY -- CARD REJECTED ' TO RR-PREFIX
0640          PERFORM  SRAE-FORMAT-EDLIST THRU SRAE-EXIT
0650          MOVE  SPACES TO CARD-FORMS
0660          GO TO ABAA-READ-CARDS.
0670      MOVE  SPACES TO RR-PREFIX.
0680      MOVE  0 TO CHI, EFI.
0690*     CHECK THE TYPE OF CARD ...1,2,3,4
0700*
0710 AJAB-PROCESS-CARD-TYPE.
0720      GO  TO  ABAB-CARD-TYPE-ONE, ABAB-CARD-TYPE-TWO
0730              ADBA-CARD-TYPE-THREE, ACAA-CARD-TYPE-FOUR
0740              ADAA-CARD-TYPE-FIVE
```

ORIGINAL INPUT

THIS EXAMPLE SHOWS ONE FACET OF THE STRUCTURING ENGINE: THE ABILITY TO REVEAL THE CONVERGENCE OF PROGRAM LEGS. THIS EXAMPLE ALSO HIGHLIGHTS LOGIC DEFICIENCIES.

THINGS NOT SHOWN DUE TO SPACE LIMITATIONS:

THE ENGINE.

CLEANS UP	-REMOVES ALTERS
LANGUAGE	-ELIMINATES PERFORM THRU OVERLAP
	-REDUCES GO TOS
	-INCREASES PERFORMS
	-CONVERTS NOTES TO COMMENTS
	-ELIMINATES DROP THRU CONFUSION
	-REMOVES DEAD CODE
STRUCTURING	-ISOLATES CONTROL HIERARCHY
	-HIGHLIGHTS LOOPING CONDITIONS
	-BOUNDS ACTION MODULES
	-PHYSICALLY GROUPS AND STANDARDIZES ALL I/O
	-CONSOLIDATES ALL PROGRAM TERMINATION TO A SINGLE GOBACK
	-DOES NOT REMOVE LOGIC ERRORS

GO TO ... DEPENDING
WITH NO EXPLICIT FALL THRU

```
                    12.43.15        MAR  4,1981

059700          MOVE SPACES                 TO  RR-PREFIX
059800          MOVE 0                      TO  CHI
059900                                          EFI
060000          PERFORM Z0210-PROCESS-CARD-TYPE.
060100*
060200 Z0210-PROCESS-CARD-TYPE.
060300      COMPUTE SV002-SR-SWITCH = CF-NO-NUM
060400      IF  (SV002-SR-SWITCH = 001)
060500          PERFORM Z0220-CARD-TYPE-ONE
060600          PERFORM Z0230-CHECK-SUBGRP
060700          PERFORM Z0240-CHECK-DIVISION
060800          PERFORM Z0250-CHECK-SUBDIV
060900          PERFORM Z0260-CHECK-BRANCH
061000          PERFORM Z0270-CHECK-USAGE
061100          PERFORM Z0280-CHECK-NAME
061200          PERFORM Z0290-CHECK-ADDRESS
061300          PERFORM Z0300-ONE-END
061400      ELSE
061500      IF  (SV002-SR-SWITCH = 002)
061600          PERFORM Z0310-CARD-TYPE-TWO
061700          PERFORM Z0320-CHECK-COUNTY
061800          PERFORM Z0330-TWO-END
061900      ELSE
062000      IF  (SV002-SR-SWITCH = 003)
062100          MOVE '3'                    TO  RR-SUFFIX
062200          PERFORM Z0350-CHECK-DEDUCTIONS
062300          PERFORM Z0360-CHECK-FREIGHT
062400          PERFORM Z0540-THREE-END
062500      ELSE
062600      IF  (SV002-SR-SWITCH = 004)
062700          PERFORM Z0550-CARD-TYPE-FOUR
062800          PERFORM Z0610-CHECK-ODO-DATE
062900          PERFORM Z0700-FOUR-END
063000      ELSE
063100      IF  (SV002-SR-SWITCH = 005)
063200          PERFORM Z0710-CARD-TYPE-FIVE
063300          PERFORM Z0730-FIVE-END
063400      ELSE
063500          PERFORM Z0220-CARD-TYPE-ONE
063600          PERFORM Z0230-CHECK-SUBGRP
063700          PERFORM Z0240-CHECK-DIVISION
063800          PERFORM Z0250-CHECK-SUBDIV
063900          PERFORM Z0260-CHECK-BRANCH
064000          PERFORM Z0270-CHECK-USAGE
064100          PERFORM Z0280-CHECK-NAME
064200          PERFORM Z0290-CHECK-ADDRESS
064300          PERFORM Z0300-ONE-END.
064400*
064500 Z0220-CARD-TYPE-ONE.
064600      MOVE '1'                    TO  RR-SUFFIX
064700      ADD 1                       TO  CHI
064800      IF  (CFOF-GROUP = SPACES
064900          OR  CFOF-GROUP NUMERIC)
```

RESTRUCTURED OUTPUT

2ND LEVEL CONTROL LOGIC
MADE EXPLICIT

GO TO ... DEPENDING
REVEALED AS CONVERGENT CASE
STRUCTURE

NOTE THAT, IN THE ORIGINAL CODE,
THE FALL THRU ARGUMENT DROPPED
INTO THE FIRST CASE

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . CODE INPUT
. . . COBOL
FUNCTION
. TRANSFORMATION
. . INSTRUMENTATION
. . EDITING
. . TRANSLATION
. . FORMATTING
. . RESTRUCTURING
. . . CODE RESTRUCTURING
. STATIC ANALYSIS
. . ERROR CHECKING
. . STRUCTURE CHECKING
. DYNAMIC ANALYSIS
. . COVERAGE ANALYSIS
. . TRACING
OUTPUT
. USER OUTPUT
. . LISTINGS
. MACHINE OUTPUT
. . SOURCE CODE OUTPUT
. . . COBOL

IMPLEMENTATION LANGUAGE: COBOL

TOOL PORTABLE: YES, TOOL SIZE: 400K

TOOL AVAILABLE: YES, PUBLIC DOMAIN: NO

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): SERVICE CONTRACT

TOOL SUPPORTED: YES, TOOL SUPPORT: CATALYST CORPORATION

CONTACT: MICHAEL J. LYONS, CATALYST CORPORATION, 5827 WEST RACE, CHICAGO, IL, 60644, USA, 312-354-5641

PSL/PSA DEMONSTRATION

Problem Statement Language/Problem Statement Analyzer (PSL/PSA) is a language and associated processor that supports an analyst in the preparation of a requirements document or a top level design of an information processing system. PSL/PSA has been developed at the ISDOS project at the University of Michigan under the direction of Professor D. Teichroew. It is currently available on a number of large scale computers.

To use PSL/PSA, the analyst first describes a portion of the target system using the formal statements of PSL. These statements are then analyzed by PSA for syntactical correctness and are added to a data base that contains all information about the target system. At the command of the analyst, various reports are generated by PSA that describe different aspects of the target system. These three steps are repeated until all aspects of the target system are described in PSL and have been entered into the data base. PSA may also be used to assist in the generation of requirements or design documentation.

It is proposed that the demonstration follow the scenario described in the previous paragraph and be supported by a remote large scale computer. The resources that would be required to run this demonstration are a 300 or 1200 baud modem, phone line, and a line terminal. We would supply the line terminal and access to a remote large scale computer. The constraint is that in any demonstration of this type only a very small example can be input and processed and there will be no reasonable way to give a copy of the data base to the ICSC attendee, other than to give the attendee the hard copy output from the terminal.

References

1. Teichroew, Daniel, and Ernest A. Hershey III, "PSA/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems", Vol. SE-3, No. 1, IEEE Transactions on Software Engineering, January 1977.

PSL/PSA Demonstration - San Diego 1

NAME=customers-and-vendors PAGE 1 OF 1

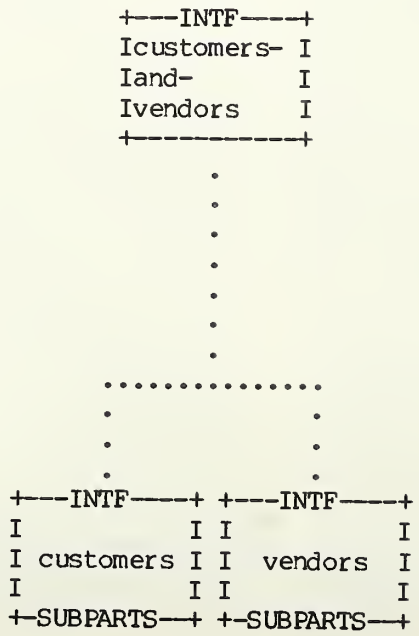
```
 +
  +---INTF----+     +---INPUT---+     +--PROCESS--+     +--OUTPUT---+     +---INTF----+
  fcustomers- I     Iinputs-fro-I     Iinventory- I     Ioutputs-to-I     Icustomers- I      Name occurs
  Iand-       I.....Im-customer-I.....Isystem     I.....I-customers-I.....Iand-       I      elsewhere.
  Ivendors    I     Is/vendors  I     I           I     I/vendors   I     Ivendors    I      See Index.
  +-----------+     +-GENERATED-+     +-RECEIVES--+     +-GENERATED-+     +-RECEIVES--+
```

PSL/PSA Demonstration - San Diego 2

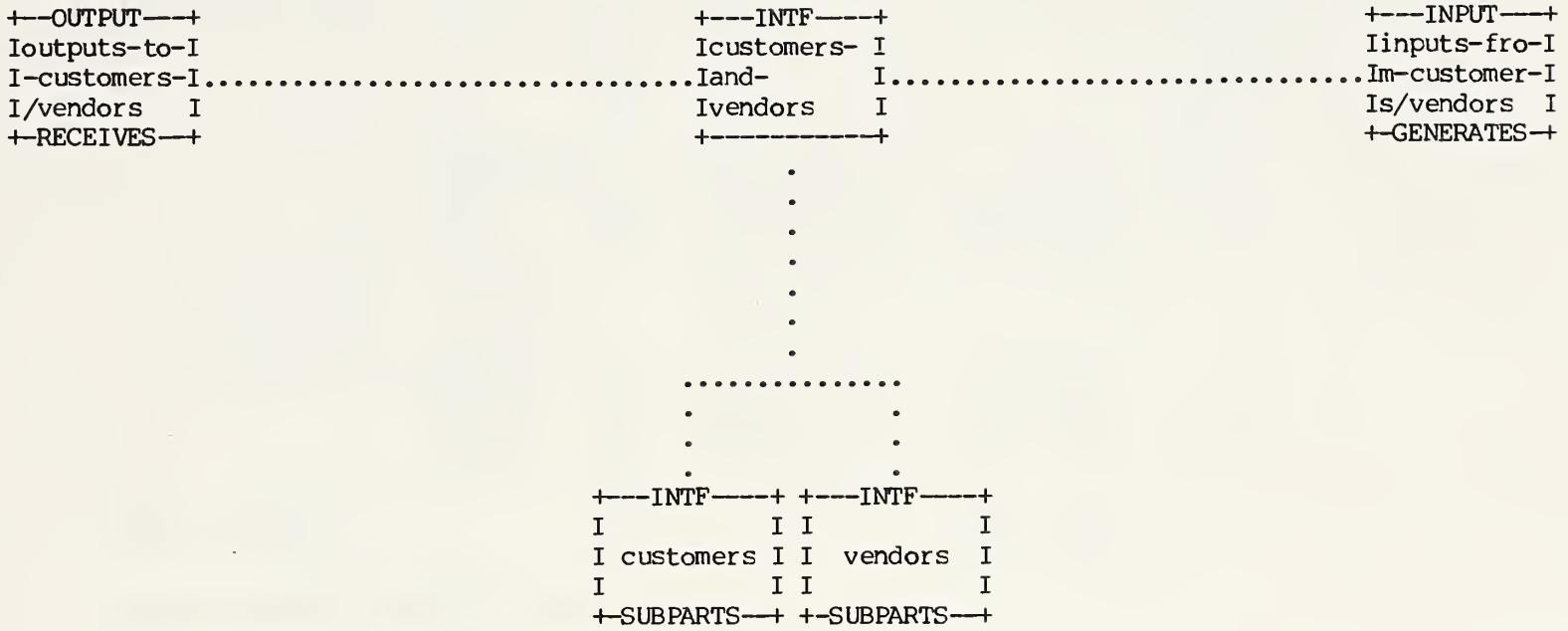
NAME=customers-and-vendors

```
+--OUTPUT---+                                +---INTF----+                                +---INPUT---+
Ioutputs-to-I                                Icustomers- I                                Iinputs-fro-I
I-customers-I................................Iand-       I................................Im-customer-I
I/vendors   I                                Ivendors    I                                Is/vendors  I
+-RECEIVES--+                                +-----------+                                +-GENERATES-+
```

NAME=customers-and-vendors

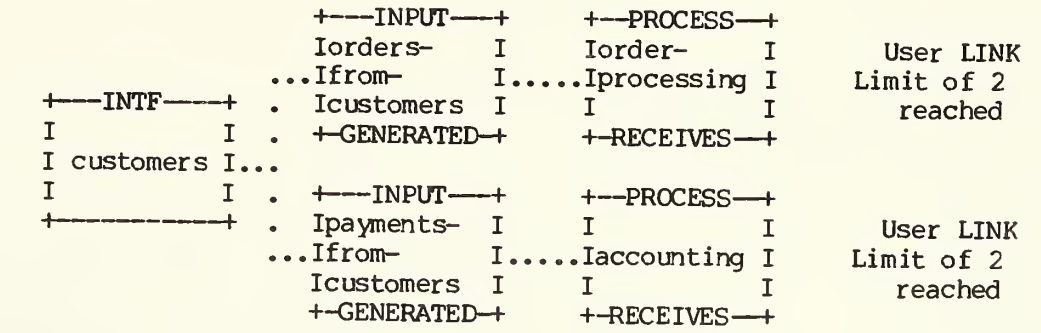


NAME=customers-and-vendors



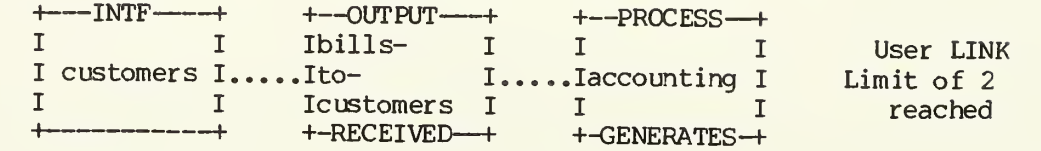
PSL/PSA Demonstration - San Diego 5

NAME=customers PAGE 1 OF 1



PSL/PSA Demonstration - San Diego 6

NAME=customers PAGE 1 OF 1



PSL/PSA Demonstration - San Diego 7

1	inputs-from-customers/vendors	INPUT
2	customers-and-vendors	INTERFACE
3	outputs-to-customers/vendors	OUTPUT
4	inventory-system	PROCESS

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . DATA INPUT
. . VHLL INPUT
. . . PSL
FUNCTION
. STATIC ANALYSIS
. . ERROR CHECKING
. . . SYNTAX CHECKING
. . COMPLETENESS CHECKING
OUTPUT
. USER OUTPUT
. . TABLES
. . LISTINGS
. . DIAGNOSTICS
. MACHINE OUTPUT
. . DATA OUTPUT

IMPLEMENTATION LANGUAGE: FORTRAN

TOOL PORTABLE: YES

TOOL AVAILABLE: YES, PUBLIC DOMAIN: NO

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): SPONSORSHIP

CONTACT: DANIEL TEICHROEW, UNIVERSITY OF MICHIGAN, ISDOS PROJECT, 231/443 WEST ENGINEERING BLDG, ANN ARBOR, MI, 48109, USA, 313-763-2238
HASAN H. SAYANI, ADVANCED SYSTEMS TECHNOLOGY CORPORATION, 9111 EDMONSTON ROAD, SUITE 302, GREENBELT, MD, 20770, USA, 301-441-9036
CYRIL P. SVOBODA, ADVANCED SYSTEMS TECHNOLOGY CORPORATION, 9111 EDMONSTON ROAD, SUITE 302, GREENBELT, MD, 20770, USA, 301-441-9036

SOFTWARE DESIGN AND DOCUMENTATION LANGUAGE (SDDL) DEMONSTRATION

SDDL is a language and associated processor that is oriented toward supporting software detailed design. The SDDL processor accepts an input of file of source statements, structures these source statements according to the general rules of structured programming, and produces various reports including a structured list of the original source statements. One of its major features is the ability to select keywords that are appropriate for the application of the user.

The proposed scenario would allow an ICSC attendee to create a very brief input file and then have this input file processed and the output returned to the attendee through a hard copy terminal. The only constraint would be the size of the input file.

References

1. Kleine, H. SDDL Reference Guide, JPL Publication 77-24

2. Callender, Clarkson, and Frasier. An Application of SDDL, JPL Report 80-16.

S D D L

SDDL STRUCTURES ILLUSTRATION

```
LINE                                                          PAGE 1
  1  PROGRAM TO EXEMPLIFY THE SDDL KEYWORDS
  2
  3       ****************************************************************
  4       *  THIS EXAMPLE ILLUSTRATES THE PROCESSOR'S REPONSE TO KEYWORD *
  5       *  STATEMENTS.                                                 *
  6       ****************************************************************
  7
  8       LOOP UNTIL FINISHED
  9           IF PROPOSITION IS TRUE
 10               DO STEP.ONE  (DEFERRED ABSTRACTION)------------->(    )
 11           ELSE
 12               DO STEP.TWO------------------------------------->(    )
 13           ENDIF
 14           IF THINGS LOOK BAD AT THIS POINT
 15       <------EXITLOOP
 16           ELSEIF THINGS LOOK DISASTROUS
 17  <-----------EXITPROGRAM
 18           ENDIF
 19       REPEAT UNTIL WE GET IT RIGHT
 20       CALL EXEMPLIFY TO DEMONSTRATE PAGE NUMBERING------------->(   1)
 21  ENDPROGRAM
```

```
LINE                                                                    PAGE     1
   4 SPECIFICATION FOR  VEEP_FUNCTIONAL_REQUIREMENTS
   5    ************************************************************************
   6    *      THIS IS A BREAKOUT OF THE FUNCTIONAL REQUIREMENTS OF THE VEHICLE  *
   7    * ECONOMY AND EMMISSIONS PERFORMANCE SIMULATION PROGRAM(VEEP).  DETAILED *
   8    * EXPANSION OF THE ITEMS LISTED BELOW MAY BE FOUND ON THE PAGES WHOSE    *
   9    * NUMBERS ARE SHOWN ON THE RIGHT.                                        *
  10    ************************************************************************
  11
  12    REQUIREMENT: SYNTAX_ANALYZER_AND_DICTIONARY_MANAGER--------------->(    )
  13    ++++++++++++++++++++++++++++++++++++++++++++++++++++
  14    + KEYWORDS FOR USER INTERACTIVE CONTROL OF THE MODEL +
  15    ++++++++++++++++++++++++++++++++++++++++++++++++++++
  16
  17    REQUIREMENT: DATA_BASE_MANAGEMENT_SYSTEM-------------------------->(  21)
  18    ++++++++++++++++++++++
  19    + VEHICLE COMPONENTS +
  20    + DRIVING CYCLE      +
  21    + ENVIRONMENT        +
  22    ++++++++++++++++++++++
  23
  24    REQUIREMENT: REPORT_GENERATOR------------------------------------->(    )
  25
  26    REQUIREMENT: ENVIRONMENT_SIMULATOR-------------------------------->(    )
  27
  28    REQUIREMENT: VEHICLE_SIMULATOR------------------------------------>(    )
  29    ++++++++++++++++++++++++++++++
  30    + STEADY STATE OPERATION     +
  31    + MAXIMUM PERFORMANCE        +
  32    + DRIVING CYCLE PERFORMANCE  +
  33    ++++++++++++++++++++++++++++++
  34
  35 END_SPECIFICATION
```

```
LINE
  84    ENTITY  'ENGINE:'
  85       'NAME'                                                    (I) TEXT
  86       'TEXTUAL.DESCRIPTION'                                     (I) TEXT
  87       'MAX.HORSEPOWER'                                          (I) HP
  88       'DISPLACEMENT'                                            (I) IN**3
  89       'MOMENT.OF.INERTIA'                                       (I) LB*INCH**2
  90       'MAX.RPM'                                                 (I) REV/MIN
  91       'IDLE.RPM'                                                (I) REV/MIN
  92       'IDLE.FUEL.FLOW'                                          (I) LB/HOUR
  93       'OUTPUT.REDUCTION.GEAR'                                   (I) UNITLESS
  94       'STEADY.STATE.TEMP'                                       (I) DEG F
  95       'TEST.FUEL.DENSITY'                                       (I) LB/GAL
  96       'TEST.FUEL.HEAT.VALUE'                                    (I) BTU/LB
  97       'TEST.DRY.BULB.TEMP'                                      (I) DEG F
  98       'TEST.RELATIVE.HUMIDITY'                                  (I) DEG F
  99       'TEST.BAROMETRIC.PRESSURE'                                (I) IN-HG
 100       'MAX.TORQUE.BOUNDARY'  (VECTOR)                           (I) LB*FT VS RPM
 101       'MIN.TORQUE.BOUNDARY'  (VECTOR)                           (I) LB*FT VS RPM
 102       'MAX.TORQUE'                                              (C) LB*FT (HIGHEST BOUNDARY VALUE)
 104       'MAX.TORQUE.RPM'                                          (C) RPM
 105       SET 'DYNO.MAP' RANKED BY LOW RPM VALUE                                                    X2
 106          'NO.OF.VALUES'    (AT EACH POINT OF THE DYNO MAP)      (I) INTEGER
 107          'VALUE.NAMES'          (VECTOR)                       (I) ALPHA
 108          'STARTUP.LOSSES'       (VECTOR)                       (I) UNITS SPECIFIED BY USER
 109          ENTITY 'RPM.SETTING:'
 110             'RPM.VALUE'
 111             SET 'TEST.POINTS'                                  (I)
 112                ENTITY 'POINT:'
 113                   'PCT.PEDAL'
 114                   'TORQUE'                                     (I) PCT
 115                   'VACUUM.VALUE'                               (I) LB*FT
 116                   'FUEL.FLOW.RATE'                             (I) IN-HG
 117                   'HC.EMISSION.INDEX'                          (I) LB/HOUR
 118                   'CO.EMISSION.INDEX'                          (I) G/KG FUEL
 119                   'NOX.VALUE'                                  (I) G/KG FUEL
 121       SET 'OTHER.LOSSES'                                       (I)
 122          (SEE CHASSIS: OTHER.LOSSES DATA STRUCTURE)                                             !GJK!
```

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . VHLL INPUT
. . . SDDL
FUNCTION
. TRANSFORMATION
. . FORMATTING
. STATIC ANALYSIS
. . SCANNING
. . . KEY WORD SCANNING
. . . STRUCTURE SCANNING
OUTPUT
. USER OUTPUT
. . LISTINGS
. . . STRUCTURED LISTINGS

IMPLEMENTATION LANGUAGE: PASCAL

TOOL PORTABLE: YES

TOOL AVAILABLE: YES, PUBLIC DOMAIN: YES

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): AVAILABLE FROM COSMIC

TOOL SUPPORTED: NO

CONTACT: H. KLEINE, JET PROPULSION LABORATORY, 4800 OAK GROVE DRIVE, PASADENA, CA, 91109, USA, 213-354-3655

SLIM

A Quantitative Tool for Software Cost and Schedule Estimation

A Demonstration of a Software Management Tool

Lawrence H. Putnam

1. Introduction

Software development has been characterized by large cost overruns and severe schedule slippages. How much will it cost? How long will it take? How many people will it take? what are the chances of the system being operational on time? What trade-off options do I have? Is the program sponsor or contractor underestimating or overestimating time, money, effort and machine time? Managers need quantitative answers to these questions.

2. Summary of SLIM

SLIM (Software Life Cycle Management) is a tool for effectively managing software development. It is a capital budgeting and strategic planning vehicle for the software life cycle. Using the PERT algorithm, linear programming, Monte Carlo simulation and sensitivity profiling techniques, SLIM provides accurate cost, time, personnel and machine projections for developing software systems. SLIM identifies limiting constraints that can block development plans. Confidence levels and risk factors are calculated to provide the manager with the hard data needed

to make decisions on cost, schedule, effort, manloading and cashflow. SLIM's accuracy has been validated for over 400 systems -- business, real time, military, telecommunications, scientific, command and control, operating systems, process control, and microprocessor software developments. SLIM is available through the timesharing facilities of American Management Systems' AMShare service. It is accessible nationwide and in many overseas locations via GTE Telenet. SLIM is also available as an in-house system running on the HP-85 personal computer in a turn-key version.

3. Scenario of SLIM Demonstration

The SLIM demonstration will be done in two parts. A brief narrative description of the software life cycle cost methodology will be given. A description of the mechanics of the computerized SLIM implementation will be made. The second part will be a demonstration of how to run SLIM, both on the timesharing computer and on the HP-85 personal computer. Questions from the audience will be welcome and members of the audience will be encouraged to run the systems themselves. The demonstration will take about one hour. It can be repeated as often as necessary. About 10 people can be handled in each demonstration group.

4. SLIM Literature

A. References on the Methodology.

Putnam, Lawrence H., Progress in Modelling the Software Life Cycle in a Phenomenological Way to Obtain Engineering Quality Estimates and Dynamic Control of the Process, Proceedings of the Second Software Life Cycle Management Workshop, Atlanta, August 21-22, 1978, IEEE Computer Society Publication, No. 78CH 1390-4C, pp. 105-128.

Putnam, Lawrence H., Example of an Early Sizing, Cost and Schedule Estimate for an Application Software System, Proceedings IEEE Computer Society COMPSAC II, Nov. 14-16, 1978, Chicago.

Putnam, Lawrence H., Software Costing and Life Cycle Control, Workshop on Quantitative Software Models for Reliability, Complexity and Costs: An Assessment of the State of the Art, Kiamesha Lake, N.Y., Sept 9-11, 1979, IEEE Catalog No. THoo 67-9, pp. 20-31.

Putnam, Lawrence H. and Fitzsimmons, Ann, Estimating Software Costs, DATAMATION, Sept 1979, pp. 189-198, Oct 1979, pp. 171-178, Nov 1979, pp. 137-140.

Putnam, Lawrence H., A General Empirical Solution to the Macro Software Sizing and Estimating Problem, IEEE Transactions on Software Engineering, Vol. SE-4, No. 4 July 1978, pp. 345-361.

B. Specific references on SLIM

* SLIM product description
* Sample output for a large system and a small system
* SLIM characteristics profile
* SLIM User's Guide for DECsystem 20
* SLIM User's Guide for HP-85

5. Location

Station 8, Tuesday, March 10 from 9:00 AM to 6:00 PM

6. The Demonstrator

Lawrence H. Putnam, President of Quantitative Software Management, Inc., 1057 Waverley Way, McLean, VA 22101. Developer of the Software Lifecycle Methodology and SLIM. He is the author of numerous articles on software cost estimating and life cycle control. He is a frequent speaker on this subject in the United States and abroad. Mr. Putnam holds a BS degree from the US Military Academy and MS-Physics from the US Naval Postgraduate School. He was elected to Sigma Xi in 1962.

****
SUMMARY OF INPUTS
****
SMALL TELECOM SYSTEM
24 JUL 81
13:07

PROJECT START: 0181

* COST ELEMENTS *

COST/MY: ($)	50000
σ(COST/MY):	5000
INFLATION RATE:	.085

* ENVIRONMENT *

ONLINE DEV:	1.00
HOL USAGE:	1.00
DEVELOPMENT TIME:	.80
PRODUCTION TIME:	.20
DBMS:	0.00
REPORT WRITER:	0.00
LANGUAGE:	PASCAL

* SYSTEM *

TYPE: TELECOMMUNICATION & MESSAGE SWITCHING

REAL-TIME CODE:	.15
LEVEL:	3
UTILIZATION:	.60

* MODERN PROGRAMMING PRACTICES *

STRUCTURED PROGRAMMING:	> 75%
DESIGN/CODE INSPECTION:	25-75%
TOP-DOWN DESIGN:	> 75%
CHIEF PROG TEAM USAGE:	< 25%

* EXPERIENCE *

OVERALL:	EXTENSIVE
SYSTEM TYPE:	EXTENSIVE
LANGUAGE:	MINIMAL
HARDWARE:	EXTENSIVE

* TECHNOLOGY *

FACTOR:	0
(ADJUSTED):	9

* SIZE *

LOWEST	10000
HIGHEST	30000

****
SIMULATION
****
SMALL TELECOM SYSTEM
24 JUL 81
13:08

	MEAN	STD DEV
SYSTEM SIZE		
(STMTS)	20000	3333
MIN DEV TIME		
(MONTHS)	13.4	1.7
DEV EFF		
(MM)	85.1	32.3
DEV COST (X 1000 $)		
(UNINFLATED)	360	148
(INFLATED)	377	157

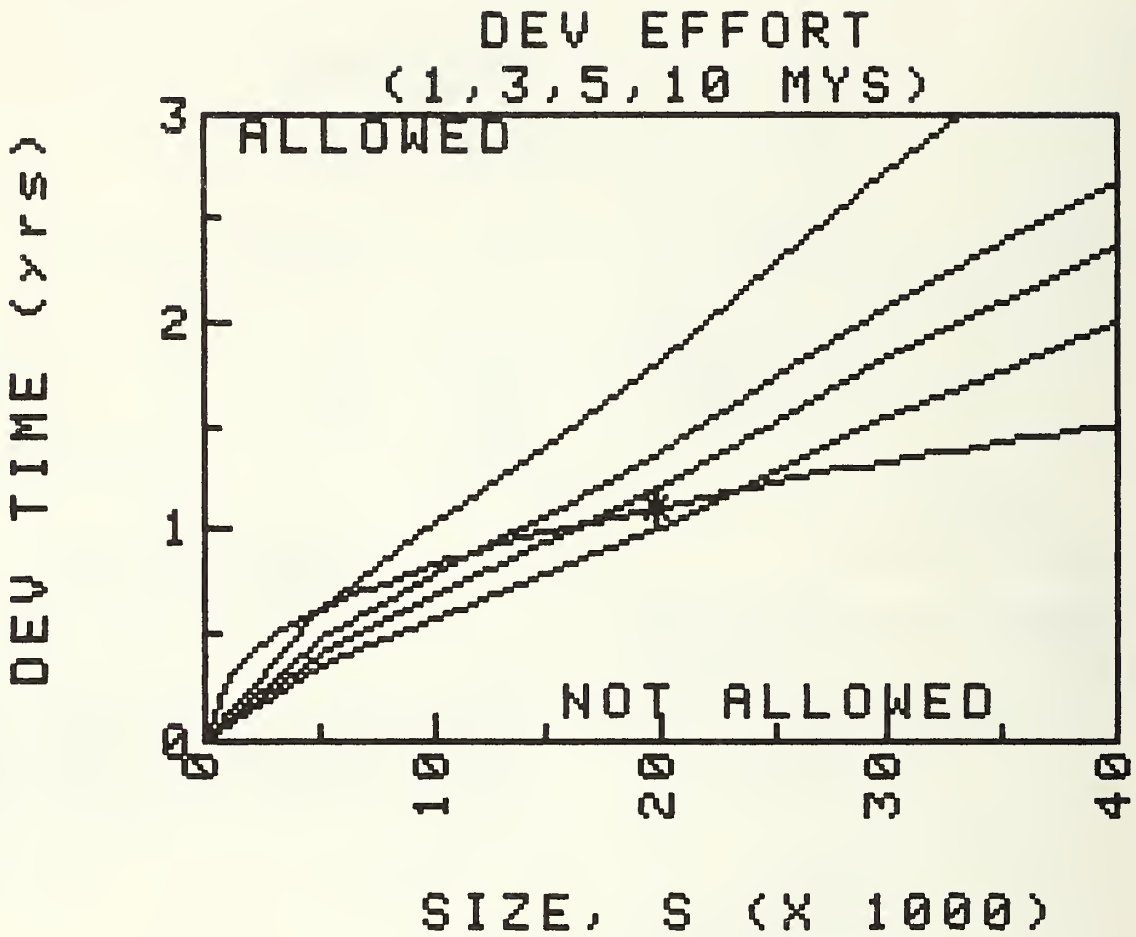
SENSITIVITY PROFILE

FOR MINIMUM TIME SOLUTION

- 3 σ	10000 STMTS 9.9 MONTHS 29 MANMONTHS 121391 $
- 1 σ	16667 STMTS 12.4 MONTHS 56 MANMONTHS 234111 $
MOST LIKELY	20000 STMTS 13.4 MONTHS 85 MANMONTHS 359897 $
+ 1 σ	23333 STMTS 14.3 MONTHS 122 MANMONTHS 507120 $
+ 3 σ	30000 STMTS 15.9 MONTHS 214 MANMONTHS 890902 $

CONSISTENCY CHECK
WITH INDEPENDENT DATA BASE

85	MM	IN NORMAL RANGE
13.4	MONTHS	IN NORMAL RANGE
6	AVG MNPWR	IN NORMAL RANGE
235	LINES/MM	IN NORMAL RANGE


DEV EFFORT
(1,3,5,10 MYS)
ALLOWED
NOT ALLOWED
DEV TIME (yrs)
0
1
2
3
0
10
20
30
40
SIZE, S (X 1000)

****
DESIGN TO COST
****

THE BEST ESTIMATES OF THE MINIMUM TIME AND CORRESPONDING EFFORT AND COST ARE:

13.41 MONTHS
85 MANMONTHS
360 (X 1000 $)

ENTER DESIRED DEV EFF IN MM
?
50

NEW DEVELOPMENT TIME
SMALL TELECOM SYSTEM

	MEAN	σ
NEW DEV TIME (MONTHS)	15.16	1.88
NEW DEV EFFORT (MANMONTHS)	50	19
NEW DEV COST (X 1000 $)	208	86

YOUR FILE IS NOW UPDATED WITH THESE NEW PARAMETERS. RUN MANLOADING & CASHFLOW TO SEE HOW THESE SAVINGS CAN BE REALIZED.

CONSISTENCY CHECK
WITH INDEPENDENT DATA BASE

50 MM	IN NORMAL RANGE
15.2 MONTHS	IN NORMAL RANGE
3 AVG MNPWR	IN NORMAL RANGE
400 LINES/MM	IN NORMAL RANGE

****
MANLOADING
****
SMALL TELECOM SYSTEM

THE TABLE BELOW SHOWS THE MEAN PROJECTED EFFORT (AND STANDARD DEVIATION) REQUIRED FOR DEVELOPMENT. THESE VALUES ARE BASED ON A DEV TIME OF 15.2 MONTHS AND A TOTAL DEV EFFORT OF 50.0 MANMONTHS.

STAFFING PLAN

MONTH	PPL	σ	CUM MM	σ
JAN 81	1	.5	1	
FEB 81	2	1.3	2	1
MAR 81	3	2.0	5	2
APR 81	4	2.6	9	3
MAY 81	4	2.9	13	5
JUN 81	5	3.1	18	7
JUL 81	5	3.1	23	9
AUG 81	5	3.0	27	10
SEP 81	5	2.8	32	12
OCT 81	4	2.5	36	14
NOV 81	4	2.1	40	15
DEC 81	3	1.8	43	16
JAN 82	3	1.5	46	17
FEB 82	2	1.2	48	18
MAR 82	2	.9	50	19
APR 82	1	.3	50	19

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . DATA INPUT
FUNCTION
. STATIC ANALYSIS
. . COST ESTIMATION
. . SCHEDULING
. . . TIME SCHEDULING
. . . PERSONNEL SCHEDULING
. DYNAMIC ANALYSIS
. . SIMULATION
. . . MONTE CARLO SIMULATION
. . LINEAR PROGRAMMING
OUTPUT
. USER OUTPUT
. . GRAPHICS
. . . BAR CHARTS
. . . LINE GRAPHS
. . TABLES
. . . SCHEDULES
. . . RISK PROFILES

IMPLEMENTATION LANGUAGE: BASIC, FORTRAN IV

TOOL PORTABLE: YES, TOOL SIZE: 200K BYTES

COMPUTER (OTHER HARDWARE): DECSYSTEM-10/20, HP-85

TOOL AVAILABLE: YES, PUBLIC DOMAIN: NO

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): LICENSE

TOOL SUPPORTED: YES, TOOL SUPPORT: QUANTITATIVE SOFTWARE MANAGEMENT

CONTACT: LAWRENCE H. PUTNAM, QUANTITATIVE SOFTWARE MANAGEMENT, INC., 1057 WAVERLEY WAY, MCLEAN, VA, 22101, USA, 703-790-0055

PERFORMANCE ORIENTED DESIGN

- BGS Systems, Inc. -

1.0 Introduction

This session will present powerful software engineering tools for the life cycle management of system performance. The tools provide essential performance related information throughout the system development cycle. They provide feedback on areas of system performance sensitivity and on alternative strategies for eliminating performance problems.

The tools can calculate key performance variables (e.g., throughput, utilization, response time) based on the specification of a system's hardware, software and workload environments. In addition, interactive front end graphics facilitate parametric studies of system performance.

2.0 Summary of Performance Oriented Design

The Basic goals of Performance Oriented Design may be summarized as follows:

- To provide better management control during system design and implementation by enabling project maangers to define and validate performance objectives at each stage of the development process.
- To reduce total development time and cost by providing designers and implementors with early warning of upcoming performance problems and by focusing attention on critical problem areas.
- To reduce maintenance costs by providing maintenance personnel with performance related information which can be consulted when performance problems arise after the system is deployed.
- To reduce development time and cost for future systems by providing designers with performance related information about existing systems which will be of direct value in future design efforts.

3.0 Demo Scenario

The demonstration will consist of an on-line interactive session/ presentation which presents details of performance analysis throughout the development life cycle. The audience will have the opportunity to participate by posing alternative configuration and design strategies which will be addressed on-line. Hands-on user time will also be made available.

4.0 References

1. POD - A software engineering tool for life cycle management of system performance, BGS Systems, TR0014-28, March 1979.

2. POD - Preliminary User's Manual, BGS Systems, TR0014-32, April 1979.

5.0 Stations, Date and Time

Short Demo: March 10, Station 9
10:00 AM - 4:30 PM and 6:00 PM - 7:30 PM.

Extended Lecture: March 10, Council Room
4:30 PM - 6:00 PM.

6.0 The Demonstration

Allan I. Levy will represent BGS Systems, Inc., a Waltham, Massachusetts firm that is actively involved in the computer performance evaluation field and is widely recognized as an industry leader in this discipline. In its work, BGS Systems emphasizes the use of analytic models for the solution of capacity planning problems. To support this activity, it has developed expertise in related areas of performance measurement, system tuning, workload forecasting, data base design and data analysis procedures. Its research contributions in computer performance have been widely recognized, and in the past three years BGS Systems' personnel have published more than twenty-five technical papers in professional journals and conference proceedings.

```
GO
```

ASSESSING THE PERFORMANCE OF THE CURRENT DESIGN

```
*** PRINCIPAL RESULTS ***

WORKLOAD                RESPONSE TIME          THROUGHPUT      % CPU

 1 POLLING_WKL             0.01 SEC         36000. PER HOUR     7.5 %
 2 STORE_WKL               1.59 SEC          6000. PER HOUR    25.0 %
 3 FLASH_WKL               0.64 SEC           667. PER HOUR     4.2 %
 4 FORWARD_WKL             1.25 SEC          1333. PER HOUR     6.9 %
                                      TOTAL CPU UTILIZATION =  43.6 %

SET FLASH_WKL ARRIVAL_RATE = 2000

GO
```

ASSESSING THE PERFORMANCE IMPACT OF A 3-FOLD INCREASE IN FLASH MESSAGE REQUESTS

```
*** PRINCIPAL RESULTS ***

WORKLOAD                RESPONSE TIME          THROUGHPUT      % CPU

 1 POLLING_WKL             0.01 SEC         36000. PER HOUR     7.5 %
 2 STORE_WKL               3.03 SEC          6000. PER HOUR    25.0 %
 3 FLASH_WKL               0.83 SEC          2000. PER HOUR    12.5 %
 4 FORWARD_WKL             1.52 SEC          1333. PER HOUR     6.9 %
                                      TOTAL CPU UTILIZATION =  51.9 %


SET FLASH_WKL ARRIVAL_RATE = 667

SET STORE_WKL ARRIVAL_RATE = 7500

GO
```

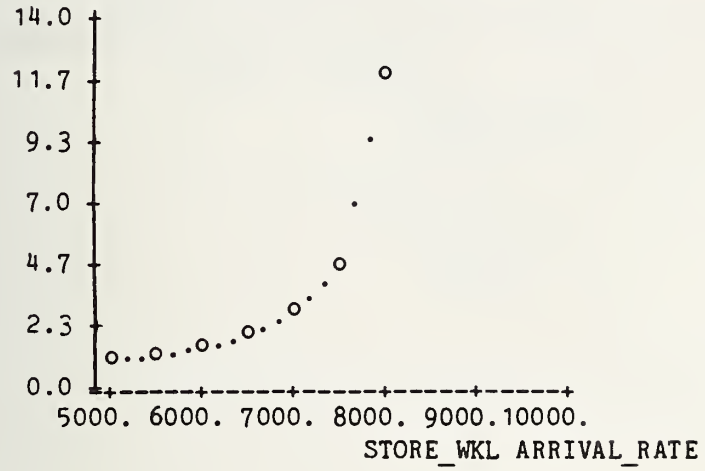
ASSESSING THE PERFORMANCE IMPACT OF A 25% INCREASE IN STORE MESSAGE REQUESTS

```
*** PRINCIPAL RESULTS ***

WORKLOAD                RESPONSE TIME          THROUGHPUT      % CPU

 1 POLLING_WKL             0.01 SEC         36000. PER HOUR     7.5 %
 2 STORE_WKL               4.61 SEC          7500. PER HOUR    31.3 %
 3 FLASH_WKL               0.66 SEC           667. PER HOUR     4.2 %
 4 FORWARD_WKL             1.28 SEC          1333. PER HOUR     6.9 %
                                      TOTAL CPU UTILIZATION =  49.9 %
```

DGRAPH 1

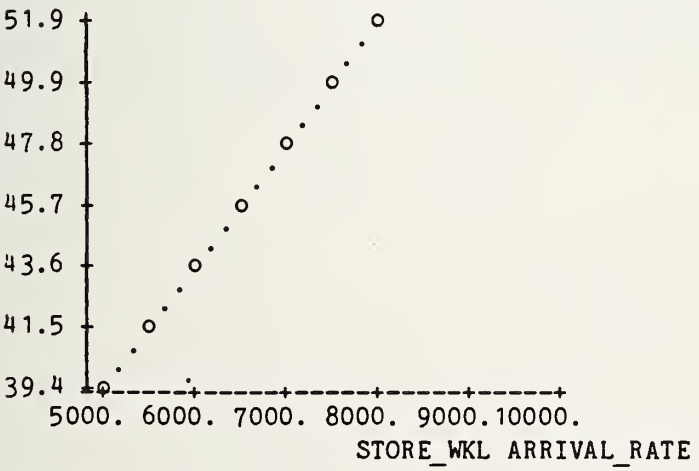
25-Nov-79 15:52
STORE_WKL RESPONSE_TIME

RESULTS OF THE PARAMETRIC STUDY PRESENTED GRAPHICALLY



DGRAPH 2

25-Nov-79 15:54
TOTAL CENTRAL_PROCESSOR UTIL



```
VARY STORE_WKL ARRIVAL_RATE

FROM 5000 TO 10000 BY 500

STOPIF STORE_WKL RT > 8

OR TOTAL CENTRAL_PROCESSOR UTIL > 85%

SHOW
STORE_WKL RT
TOTAL CENTRAL_PROCESSOR UTIL
END

RUN
```

PARAMETRIC STUDY TO DETERMINE THE PERFORMANCE SENSITIVITY OF THE DESIGN TO A RANGE OF STORE MESSAGE ARRIVAL RATES

* PAGE 1 OF 2

	STORE_WKL ARRIVAL_RATE			
	5000.0	5500.0	6000.0	6500.0
STORE_WKL RESPONSE_TIME	1.1	1.3	1.6	2.1
TOTAL CENTRAL_PROCESSOR UTIL	39.4	41.5	43.6	45.7

* PAGE 2 OF 2

	STORE_WKL ARRIVAL_RATE		
	7000.0	7500.0	8000.0
STORE_WKL RESPONSE_TIME	2.9	4.6	12.0
TOTAL CENTRAL_PROCESSOR UTIL	47.8	49.9	51.9

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . VHLL INPUT
. . . SYSTEM SPECIFICATION
FUNCTION
. DYNAMIC ANALYSIS
. . SIMULATION
. . TUNING
OUTPUT
. USER OUTPUT
. . GRAPHICS
. . TABLES

IMPLEMENTATION LANGUAGE: FORTRAN

TOOL PORTABLE: YES

TOOL AVAILABLE: YES, PUBLIC DOMAIN: NO

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): DETAILS FROM BGS SYSTEMS

TOOL SUPPORTED: YES

CONTACT: ALLAN LEVY, BGS SYSTEMS, INC., 47Ø TOTTEN POND ROAD, WALTHAM, MA, Ø2254, USA, 617-89Ø-ØØØØ

Programmer Workbench Tools on VAX/VMS

Heinz Lycklama

INTERACTIVE Systems Corporation

Software developers need good tools to improve their productivity. One such set of tools has been available for some time now with the Programmer's Workbench version of UNIX * running on the DEC PDP11 series of computers. The recent introduction of the DEC VAX computers with the VMS operating system left something to be desired in the area of software tools. A rich set of languages have been and are being developed under the VMS operating system, but there was nothing equivalent to the PWB tools. INTERACTIVE Systems Corporation has recently introduced the PWB tools on the VAX/VMS system, thereby improving the productivity of software engineers on this machine significantly. The tools have been designed to work effectively in the VMS environment.

A programmer needs a number of tools to accomplish the design and implementation of the end product:

editors
compilers
source code control system
program maintenance system
documentation system

This demonstration will concentrate on how a general-purpose source code control system and a program maintenance system can be used to advantage in an operating system environment for which it was not initially designed.

The Source Code Control System (SCCS) is a system for controlling changes to files of text such as the source code and documentation of software systems. It provides facilities for storing, updating, and retrieving any version of a file of text, for controlling updating privileges to that file, for identifying the version of a retrieved file, and for recording who made each change, when and where it was made, and why. SCCS is a collection of less than a dozen programs or commands available to the user. The SCCS commands consist of:

1. admin - administer SCCS files
2. chghist - change history entry of an SCCS delta
3. comb - combine SCCS deltas
4. delta - make an SCCS delta
5. get - get generation from SCCS file
6. prt - print SCCS file
7. rmdel - remove a delta from an SCCS file
8. sccsdiff - compare two versions of an SCCS file
9. what - identify version of a file

In the VAX/VMS environment, the SCCS system is sufficient for monitoring changes to source code in any language as well as documents written using the editor.

The Make facility is provided for maintaining computer programs. When dealing with a large software development project, it is common practice to break large programs into small manageable files. Each one of these pieces is likely to require a different treatment to produce the object modules

* UNIX is a Trademark of Bell Telephone Laboratories.

that need to be linked into a final executable program. In some cases, object modules are produced by using one of many compilers available. In other cases the source code may need to be run through a macro processor or even processed by one of the sophisticated program generators available, such as yacc, a compiler-compiler, or lex, a lexical analyzer generator. The final code resulting from all of the possible transformations may then need to be loaded together with certain libraries under the control of special options. Even for the very sophisticated programmer it is easy to forget which files depend on which others, which files have been modified recently, and the exact sequence of operations needed to make as well as exercise a new version of the program. The Make program automates all of the necessary steps that a programmer would have to go through to produce an up-to-date executable program, given that the graph of file dependencies is correct. However, it does not reproduce those object modules which are already up-to-date, in the process. On the VAX, the Make program has been modified to fit into the VMS environment. Given a new language compiler, it is easy to specify the new 'suffix' rules with the make facility.

Dr. Heinz Lycklama, Vice President - Technical Development

Heinz is head of the technical product development team at INTERACTIVE Systems, and in this capacity is responsible for all new product development. He holds a Ph.D. from McMaster University, and spent nine years at Bell Telephone Laboratories in research and development, both at Murray Hill and Holmdel, before joining INTERACTIVE Systems in 1978. Eight of his years at Bell were spent in operating systems research, and during much of that time he was associated with the UNIX operating system. His last position at Bell Labs was as a supervisor of a group responsible for designing a message switching and data entry system in a data communications network.

Heinz has made many contributions to operating systems research: He participated in design and implementation of a virtual memory operating system for a Honeywell DDP-516 computer; added asynchronous I/O and large contiguous file system to an early version of UNIX; codesigned and built the MERT operating system for the DEC PDP-11 computers; codesigned a satellite processor system to support a large number of micro's and mini's attached to a central PDP-11 host machine; developed a single-user UNIX system for the LSI-11 microcomputer; developed the Mini-UNIX system for small PDP-11 computers; and developed a number of interactive programs suitable for CRT terminals and intelligent terminals. Much of this work has been published in the Bell System Technical Journal and other professional publications.

At INTERACTIVE Heinz led the team which developed the UNIX system on the VAX/VMS operating system for the DEC 11/780 computer.

```
s.n1.c:

Checked out for editing --
        1.14 1.15 jim 81/03/03 13:16:03

D 1.14  80/12/19 02:53:06       jim     14      13      00001/00001/01295
Avoid calling rbf0 if ip == -1; caused .rd to crash on the VAX

D 1.13  80/09/23 16:25:56       hal     13      12      00002/00001/01294
        cut down on ttyn calls by allowing a gtty failure in place of
ttyn returning 'x'.

D 1.12  80/09/16 02:57:53       steve   12      11      00015/00014/01280
Fix bug which caused terminal to be made unwritable even if
output was not to the terminal.  Save unnecessary ttyn's.

D 1.11  80/02/25 21:35:44       heinz   11      10      00009/00001/01285
Changes for CAT/8.

D 1.10  79/12/27 08:42:57       heinz   10      9       00002/00003/01284
Portability changes.

D 1.9   79/07/27 15:57:41       hal     9       8       00005/00000/01282
fix vax .ev bug.

D 1.8   79/07/19 21:27:32       hal     8       7       00303/00037/00979
add save-restore option.

D 1.7   79/07/10 10:21:10       hal     7       6       00117/00046/00899
add jim's vax changes

D 1.6   79/04/12 14:23:19       jim     6       5       00011/00007/00934
clean up gtty/stty so there is a set for both input and output

D 1.5   79/04/11 19:07:15       jim     5       4       00002/00002/00939
Fixed stty bug that messed up Diablo/Qumes by
setting them to the speed of the input terminal rather than
the output device.

D 1.4   79/01/08 03:02:05       jim     4       3       00005/00000/00936
added -O flag for absolute page numbers

D 1.3   78/12/01 09:18:41       jim     3       2       00002/00003/00934
fix cnts underscore

D 1.2   78/11/30 03:27:58       jim     2       1       00000/00006/00937
gets rid of spurious "intermediate language" tap

D 1.1   78/11/29 17:58:47       mike    1       0       00943/00000/00000
```

```
Users allowed to make deltas --
        steve
        heinz
        hal
        jim

Flags --
        none

Description --
        This file is the Version 7 Phototypesetter NROFF source as
        received from Bell Labs, with slight modifications. The



        SCCS'ing of these files is prepartory to making changes
        required by our support of Davis, Polk, Wardwell and the
        -mw macro package.


    s.n3.c:

    D 1.10      80/12/19 03:02:48       jim     10      9       00004/00012/00858
    eliminate UNSIGNED since unsigned now works on VAX;
    fix bug, cstate -> cstate[index]

    D 1.9       80/04/28 21:54:43       heinz   9       8       00001/00001/00869
    Changes made for portability to reflect changes in VAX C compiler.

    D 1.8       79/12/27 08:45:52       heinz   8       7       00010/00008/00860
    Portability changes.

    D 1.7       79/07/27 18:40:48       hal     7       6       00000/00002/00868
    make size of cache be settable from makefile.

    D 1.6       79/07/27 16:29:26       hal     6       5       00002/00000/00868
    make UNSIGNED defined for -11 as well as vax.

    D 1.5       79/07/27 15:58:40       hal     5       4       00021/00004/00847
    fix vax .ev bug; correct problem due to vax C compiler bug:
    unsigneds lose their left halves at random times.

    D 1.4       79/07/19 21:29:41       hal     4       3       00181/00016/00670
    cache macros.

    D 1.3       79/07/10 10:25:49       hal     3       2       00065/00024/00621
    add jim's vax changes

    D 1.2       78/12/01 09:38:38       jim     2       1       00000/00000/00645
    fix cnts underscore

    D 1.1       78/11/29 17:59:03       mike    1       0       00645/00000/00000
```

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . DATA INPUT
. . CODE INPUT
FUNCTION
. TRANSFORMATION
. . EDITING
. STATIC ANALYSIS
. . MANAGEMENT
. . . CONFIGURATION MANAGEMENT
. . COMPARISON
. . COMPLETENESS CHECKING
OUTPUT
. USER OUTPUT
. . TABLES
. . DIAGNOSTICS
. MACHINE OUTPUT
. . OBJECT CODE OUTPUT
. . DATA OUTPUT

IMPLEMENTATION LANGUAGE: C

TOOL PORTABLE: PARTIAL

COMPUTER (OTHER HARDWARE): VAX 11/780, PDP 11, ONYX

OS (OTHER SOFTWARE): UNIX, VMS

TOOL AVAILABLE: YES, PUBLIC DOMAIN: NO

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): LICENSE

TOOL SUPPORTED: YES, TOOL SUPPORT: INTERACTIVE SYSTEMS CORP.

CONTACT: HEINZ LYCKLAMA, INTERACTIVE SYSTEMS CORPORATION, 1212 SEVENTH ST., SANTA MONICA, CA, 90401, USA, 213-450-8363

Affirm
A Specification and Verification System

A Software Engineering Tool Demonstration

Roddy W. Erickson, Susan L. Gerhart, Stanley Lee, and David H. Thompson

1. A Brief Synopsis

What is Affirm?

Affirm is an experimental system for the specification of *abstract data types* and *algorithms*, and the verification of their properties. The system uses the *algebraic axiomatic* specification technique, where the user specifies a set of *operations* and a set of *axioms* detailing how the operations work. Algorithms may be expressed either in terms of abstract data types, or in a Pascal-like language, or both. General systems, such as communication networks, may also be modeled using the algebraic axiomatic method.

The heart of the system is a "natural deduction" theorem prover for interactively proving properties of specifications and algorithms. Other features include a library of data types, and extensive user interface facilities.

Experience with ***Affirm*** includes extensive experimentation with data type specifications, verification of small algorithms, the specification and partial proof of a large file-updating module, and the proof of high-level properties of protocols, Security kernels, and distributed file systems.

How do I use it?

A typical session with the system consists of signing on, reading in type specifications, proposing theorems to be proved, and interactively developing proofs. Finally, the user can freeze the system state for further use, and print out a transcript of the session for further study.

What's an abstract data type?

An abstract data type consists of a set of *operations* and a set of *axioms*. The axioms define the operations by describing their *observable behavior* (i.e., the axioms tell what the operations do, but not how they're implemented).

In proving theorems using ***Affirm***, the axioms of data types are used to rewrite expressions to simpler forms: the axioms become part of the *simplification* process.

What's an algorithm? Is it a program?

Affirm deals with specifications and *algorithms*. We say algorithm rather than program for several reasons. First, the days of verifying programs *only when they're all finished being coded* are past: now, verification of the *intended* code is performed--the *specification* of the problem is somehow validated. Second, the system accepts "programs" in a language close to Pascal, but it isn't Pascal. No compiler for it exists. Other verification systems address real programming languages; we simply choose to emphasize *specifications*.

What sorts of things do you prove using Affirm?

We prove *properties* of *specifications*. If the specification is of an abstract data type, such as for example *Sequence*, we might prove some useful property about sequences. If the specification is of a protocol, we might state and prove that the protocol doesn't deadlock, or that it will deliver the right messages in the right order. If the specification is of an algorithm for sorting integers, we might prove that it indeed sorts the same integers handed to it.

How do you prove properties in Affirm?

The theorem prover in ***Affirm*** provides a large number of commands that help break a complex expression into a series of smaller ones. And the theorem prover uses the axioms of the defined data types to rewrite pieces of an expression into simpler forms.

The system's theorem prover is an interactive, natural-deduction prover. The user guides the system through the steps of a proof; the system performs the bookkeeping, keeps the user honest, and performs much of the simplifications along the way. But the prover does not generate the proof by itself; the human user does. The sequence of steps leading to a proof starts with the user stating a property to be proved, in the language of first-order predicate calculus. The objective at each step is to either immediately simplify the expression to TRUE, if possible, or to break the complex expression up into a series of smaller, hopefully simpler expressions. The approach is thus "divide and conquer." The system keeps track of what's left to prove, records the steps taken so far, and helps the user move among the various expressions that must each be simplified to TRUE in order for the original expression to be proved.

Do I have to be some sort of mathematical or logic whiz to use Affirm?

It helps. But it's not necessary. We've trained people in one day to be able to use the system effectively. Much of the experience with *Affirm* was accumulated by people not associated with our project.

Can I verify my payroll system?

No. We can verify "programs" expressed in our "programming language," closely akin to Pascal. But you'd have to translate the individual routines into our language, and specify what you want to prove about them. And the verification effort for a large system would be very expensive, given current resources.

What are Affirm's strengths and weaknesses?

Strengths:

- The specification methodology is extremely general. We've described security kernels, communication protocols, data types, algorithms for numerical analysis and distributed file systems, etc., all in the same algebraic axiomatic way.
- We've emphasized the user interface--pieces of the system whose job it is to make the whole thing easier to use. Spelling correction, user profiles, and the like help the user use the system more effectively.
- Extensive documentation of the system is available. There are five volumes in the reference library, consisting of about 500 pages of material: the Reference Manual, a User's Guide, a Type Library, the Annotated Transcripts, and the Collected Papers.
- The system naturally follows modern programming methods--abstraction--though no specific style is enforced by the system.

Weaknesses:

- The specification technique doesn't handle the explicit notion of an error. In terms of a sequence, what's the first element of an empty sequence? Would a program implementing the first operation return an error, or what?
- The specification technique doesn't directly deal with notions of parallelism or concurrency.
- The theorem prover consumes computer space and computer time. Resource requirements are the major limitation of the system at this time.
- It is often necessary for the user to direct the theorem prover in smaller steps than we would like. ***Affirm*** is particularly weak in its knowledge of integers.

What exactly is the experience to date?

- With data abstraction as its main methodology, *Affirm* has been pushed heavily in specifying and verifying a number of data types, including *Sequence*, *Queue*, *Mapping* (or *Array*), *Circle*, *Set*, and *BinaryTree*.
- In the *Delta* experiment, ten algorithms were abstracted from about 1000 lines of code from a message system. It was proved that each of the procedures computed several specified functions, and that these functions satisfied an overall system requirement.
- A toy security kernel was specified, and a security property was verified, in the space of two days.
- A number of different communication protocols have been specified, and various properties of the protocols verified:
  - The Alternating Bit protocol was specified in several "layers," all the way from the high-level abstract statement that the protocol acts like a queue, through the description of a nondeterministic distributed protocol, down to the "code." We verified a number of properties stating that the various specifications were consistent with one another, and that the right messages were delivered in the right order.
  - The three-way handshake protocol (part of a real protocol used in the ARPANET) was specified, and some properties about it verified. Similar work was done for the majority consensus protocol (a concurrency control protocol for distributed databases).

2. A Scenario of an Affirm Demonstration

A demonstration of *Affirm* consists of two parts. The first part is a detailed description of an abstract data type specification. This entails describing how one specifies a data type in ***Affirm*** and what that specification means. This first part takes about 10 minutes. We will then take a small theorem and prove it interactively. This will hopefully give the audience a feeling of actual ***Affirm*** usage. The whole highly interactive demonstration will take about 30 minutes.

3. Affirm Literature

The ***Affirm*** Reference Library consists of five volumes. The Reference Manual contains a detailed discussion of the major concepts behind ***Affirm***, presented in terms of the abstract machines that form the structure of the system as seen by the user. The Users Guide is a question-and-answer dialogue detailing the whys and wherefores of specifying and proving using ***Affirm***. The Type Library documents the several abstract data type specifications developed and used by the ISI Program Verification Project, with additional commentary and alternative definitions where applicable. The Annotated Transcripts volume of the reference libary consists of a series of heavily annotated transcripts displaying ***Affirm*** in action, to be used as a sort of workbook along with the Users Guide and Reference Manual. And the Collected Papers is a collection of articles authored by members of the ISI Program Verification Project (past and present), updated with new examples from the latest version of ***Affirm***. In addition, the following papers of general interest are available in the literature:

[1] Gerhart, S. L., et al. An overview of ***Affirm***: a specification and verification system. In *Proceedings IFIP 80*, pages 343-348. Australia, October, 1980.

[2] Guttag, J. V. Notes on type abstraction. *IEEE Transactions on Software Engineering* SE-6(1):13-23, January, 1980.

[3] Musser, D. R. Abstract data type specification in the ***Affirm*** system. *IEEE Transactions on Software Engineering* SE-6(1):24-32, January, 1980.

[4] Thompson, D. H., C. A. Sunshine, R. W. Erickson, S. L. Gerhart, and D. Schwabe. *Specification and Verification of Communication Protocols in **Affirm** using State Transition Models*. ISI/RR-81-88, USC/Information Sciences Institute, February, 1981. (Also submitted for publication).

4. Station, Day, and Time

We will demonstrate ***Affirm*** at **Station 1, Wednesday, March 11,** from **11:00 am until 3:00 pm.**

5. The Demonstrators

Roddy W. Erickson

Rod designed and implemented the proof structure underlying the theorem-prover component of the ***Affirm*** system. His research interests include specification methodologies and issues of network communication.

Stanley Lee

Stan has spent the last year as one of the main users of the system. He has used ***Affirm*** in the verification of a program using rational arithmetic, and has worked on the specification and proof of high-level file update consistency properties of a distributed file system.

Susan L. Gerhart

Susan has been involved with ***Affirm*** since its inception in 1978 as the primary critical user. She has used ***Affirm*** for numerous small examples and data types and has led efforts in larger applications of the system to such areas as protocols and file updating.

David H. Thompson

David implemented much of the user-interface of the system. His research interests include issues of user habitability, both in the design process for new systems and as an add-on capability for existing systems.

6. Example Output: Proof of "nodups(dedup(s))"

The property proved here is a statement of consistency between two operations on *sequences*. *dedup* is an operation that removes duplicate elements from a sequence. *nodups* is a predicate that tests a sequence; it returns TRUE if no duplicates are present.

```
axioms
    dedup(Empty) = = Empty,
    dedup(s apr i) = = if i in s
                       then dedup(s)
                       else dedup(s) apr i;
axioms
    nodups(Empty) = = TRUE,
    nodups(s apr i) = = (nodups(s) and not (i in s));
```

Thus one property that should hold for all sequences *s* is "nodups(dedup(s)):" a sequence whose duplicates have been removed should have no duplicates. The following transcript was automatically produced by the system during the interactive proof of the above theorem. The italicized comments were added afterwards. User-typed input is display in **a large font**; system-generated output is displayed in a small font.

```
Transcript file <DTHOMPSON>TRANSCRIPT.AFFIRM.3-MAR-81.3
is open in the Affirm system <AFFIRM>AFFIRM.EXE.121
```

get the Sequence type from the library.

```
1 U:needs type sequence;
 compiled for Affirm on  3-Mar-81 11:13:17
 file created for Affirm on  3-Mar-81 11:12:55
SEQUENCECOMS
 compiled for Affirm on  3-Mar-81 11:17:00
ELEMENTCOMS
<DTHOMPSON>ELEMENT.COM.1<DTHOMPSON>SEQUENCE.COM.1
```

make the type's declarations available for use in proving. The system responds by echoing the usable declarations.

```
2 U:edit sequence;
(sequence => Sequence)
type Sequence
s, ss, s1, s2: Sequence
k, ii, i, i1, i2, j: Element
3 U:freeze demo;
Writing file... <DTHOMPSON>DEMO.EXE.1
4 U:try nodupsAfterDeduped, nodups(dedup(s));
nodupsAfterDeduped is untried.
all s (nodups(dedups(s)))
```

To prove our theorem we use induction on the structure of sequence s. *That is, we must show the theorem holds for Empty, the empty sequence, and for any sequence of the form some-sequence append-right ii, or* ss apr ii. *The Empty case falls out directly from the axioms for nodups and dedup. The general case, as usual, requires some more proof steps (commands to Affirm). IH is an abbreviation for the Induction Hypothesis; in proving the* ss apr ii *case, we can assume (as a hypothesis) that the theorem holds for sequence* ss.

```
5 U:employ Induction(s);
Case Empty: Prop(Empty) proven.
Case apr: all ss, ii (IH(ss) imp Prop(ss apr ii)) remains to be shown.
 (apr:)
all ss, ii
 (  IH(ss, 1 {nodupsAfterDeduped})
  imp if ii in ss
        then nodups(dedup(ss))
```

```
        else nodups(dedup(ss)) and ~(ii in dedup(ss)))
```

The inductive hypothesis IH.

```
6 U:print IH;
   IH(ss, 1 {nodupsAfterDeduped}) = = nodups(dedup(ss))
```

At any time during the proof we can examine a summary of proof steps performed so far; all we've done so far is to employ Induction to produce two subgoals, one of which immediately reduced to TRUE. *The arrow (->) shows our current location in the proof.*

```
7 U:print proof;
theorem nodupsAfterDeduped, nodups(dedup(s));
proof tree:
5:|  nodupsAfterDeduped
      employ Induction(s)
     Empty:
      immediate
5:   apr:
      2  cases
?->    3
```

Expand the reference to IH by replacing the reference with the definition (and then simplify).

```
8 U:invoke IH;
all ss, ii (nodups(dedup(ss)) imp ii in ss or ~(ii in dedup(ss)))
```

The form of the above expression suggests a lemma stating that if an element appears in `dedup(s)`, *it must also be in* `s`. *Several things happen next. The lemma is stated, but no name is provided (I forgot). So the system generates one for me. The profile entry AutoSearch is set, so the system automatically looks for an instantiation of the variables of the lemma--it looks for likely values in the expression being proved. In this case, the system found an instantiation that actually simplified the expression being proved to* TRUE. *The system then keeps us honest: we next have to validate the lemma.*

```
9 U:apply i in dedup(s) imp i in s;
Since you didn't supply a name, that's called lemma1OfnodupsAfterDeduped
some i, s (i in dedup(s) imp i in s)
(will search for instantiation)
1/2: (s = ss) and (i = ii)
Proved by chaining and narrowing using the substitution (s = ss) and (i = ii)
TRUE
nodupsAfterDeduped is awaiting the proof of lemma lemma1OfnodupsAfterDeduped.
Going to lemma lemma1OfnodupsAfterDeduped.
lemma1OfnodupsAfterDeduped is untried.
all i, s (i in dedup(s) imp i in s)
```

We change the name to something more mnemonic.

```
10 U:name inDedup, lemma$;
(lemma$ => lemma1OfnodupsAfterDeduped)
```

Again, induction is our main tool. The definition of the inductive hypothesis IH and the proof are displayed below.

```
11 U:employ Induction(s);
Case Empty: Prop(Empty) proven.
Case apr: all ss, ii (IH(ss) imp Prop(ss apr ii)) remains to be shown.
 (apr:)
all ss, ii, i (   IH(ss, 5 {inDedup}) and i in dedup(ss)
        imp i in ss or (i = ii))
12 U:print IH;
```

Note the universal quantifier `i` *in the definition of IH.*

```
    IH(ss, 5 {inDedup})
      = = all i (i in dedup(ss) imp i in ss)
```

When we replace the reference to the inductive hypothesis IH with its definition, The universal quantifier i becomes an existential quantifier (a rule of logic: the universal quantifier is part of the left operand of an implication). The system again looks for a likely instantiation. It finds one, and we're done. The system then automatically prints the completed proof trees.

13 U:invoke IH;
(will search for instantiation)
1/1: i' = i
Proved by chaining and narrowing using the substitution i' = i
TRUE
inDedup proved.
nodupsAfterDeduped proved.

theorem inDedup, i in dedup(s) imp i in s;
proof tree:
11:I inDedup
    employ Induction(s) {proved by Dthompson using Affirm 121 on 3-Mar-81 in
        transcript <DTHOMPSON>TRANSCRIPT-AFFIRM.3-MAR-81.5}
  Empty:
    immediate
11: apr:
    8 cases
13: 9 invoke IH
13: 10 put i' = i {search}
13:-> (proven!)

theorem nodupsAfterDeduped, nodups(dedup(s));
nodupsAfterDeduped uses inDedup!.
proof tree:
5:I nodupsAfterDeduped
    employ Induction(s) {proved by Dthompson using Affirm 121 on 3-Mar-81 in
        transcript <DTHOMPSON>TRANSCRIPT-AFFIRM.3-MAR-81.5}
  Empty:
    immediate
5: apr:
    2 cases
8:   3 invoke IH
9:   4 apply inDedup
9:   6 put (s = ss) and (i = ii) {search}
9:   (proven!)
The proof of this part is finished.
No theorems are untried.
No theorems are tried.
No theorems are awaiting lemma proof.
14 U:quit;
Save this Affirm session? no [confirm]
Type CONTINUE to return to Affirm.

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . CODE INPUT
. . VHLL INPUT
. . . ALGEBRAIC SPECIFICATIONS
FUNCTION
. STATIC ANALYSIS
. . CONSISTENCY CHECKING
. . TYPE ANALYSIS
. DYNAMIC ANALYSIS
. . ASSERTION CHECKING
. . . FORMAL PROOF OF CORRECTNESS
OUTPUT
. USER OUTPUT
. . LISTINGS
. . DIAGNOSTICS

IMPLEMENTATION LANGUAGE: LISP INTERLISP

TOOL PORTABLE: NO

COMPUTER (OTHER HARDWARE): DECSYSTEM-10/20

OS (OTHER SOFTWARE): INTERLISP

TOOL AVAILABLE: YES, PUBLIC DOMAIN: YES

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): ACCESS PREFERED OVER ARPANET

TOOL SUPPORTED: YES, TOOL SUPPORT: INFORMATION SCIENCES INSTITUTE

CONTACT: R. W. ERICKSON, INFORMATION SCIENCES INSTITUTE, 4676 ADMIRALTY WAY, MARINA DEL REY, CA, 90291, USA, 213-822-1511
S. L. GERHART, INFORMATION SCIENCES INSTITUTE, 4676 ADMIRALTY WAY, MARINA DEL REY, CA, 90291, USA, 213-822-1511

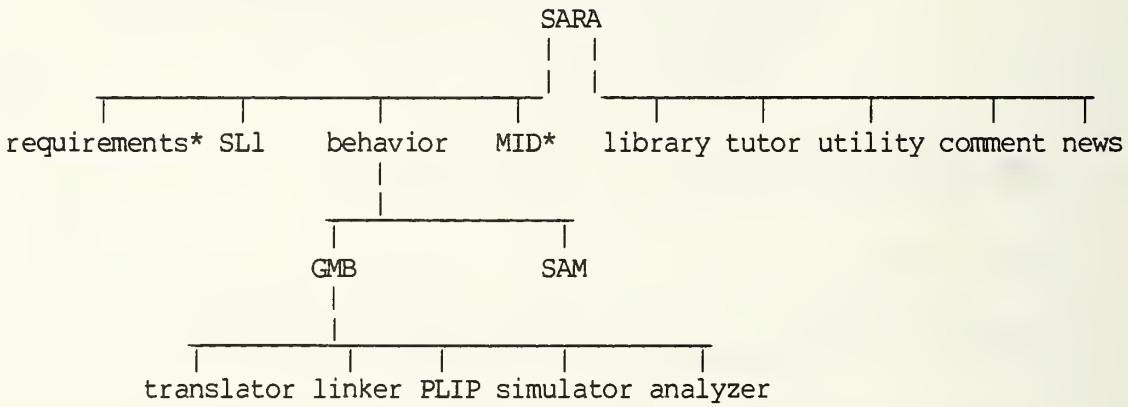
SARA as a tool for software design: Building-block modelling and composition +

Maria Heloisa Penedo
Computer Science Department
University of California, Los Angeles

1. Introduction:

SARA (System ARchitects' Apprentice) is a computer-aided design system, currently under development at UCLA, which supports a structured multi-level design methodology for the design of hardware or software systems. It comprises a number of language processors and tools for assisting designers using the SARA methodology, together with a user-interface capability for assisting designers using the SARA system. The SARA system is implemented on the MIT Multics system and is readily accessible through ARPANET or TELENET.

The hierarchy of the SARA system is illustrated below:



The leaves of the left branch of the tree are the tools supporting the methodology while the leaves of the right branch are the tools supporting the SARA system [FenR80]. The asterisks indicate that the tools have not been implemented yet.

2. Summary:

The SARA methodology is requirement driven and it supports both top-down and bottom-up design procedures. Each step of a design starts with the definition of the requirements for the system and the assumptions about the system's environment. The tools to accept and analyze requirement definitions are currently under development and are discussed in a recently completed dissertation [WinJ80].

+ This work was supported by the Department of Energy, Contract No. DE-AF03-765F0034 P.A., No. DE-AT-036, ER70214, Mod. A006.

Hardware and software systems are designed in SARA by modelling their structure and behavior. The tools SL1 (Structural Language 1) and GMB (Graph Model of Behavior) support definition of structural and behavioral models respectively. GMB allows definition of behavior in three domains: control-flow, data-flow and interpretation. The GMB Translator is the language processor for the control and data-graph definitions; PLIP is the language processor for the interpretation.

GMB models are mapped to structures and are denoted SL1-GMB models. The linking of GMBs mapped to connected structures is processed by the GMB-Linker.

The GMB Simulator provides an interactive simulation environment which permits experiments on the behavioral models. The GMB Analyzer allows designers to perform formal analysis on the GMB control-graph [RazR80a,b].

Behavioral Attributes can be associated with structures by means of SARA's Attribute-Based Model (SAM) [SamA81].

A top-down design strategy can be applied by refining structural and behavioral models of the system. A bottom-up design strategy can be applied by composing structural and behavioral models of existing building-blocks [DroJ80].

In software design, a path between modelling and code is provided by the definition of the structure of code and a mapping between the structure of the models and the code structure. The MID (Module Interface Description) tool (not yet implemented) permits these definitions [PenM80,81].

3. Scenario:

The use of the SARA tools in the design of concurrent software is illustrated by the design of a bounded buffer with asynchronous read and write operations, and by the use of two instances of the buffer system in the composition of a larger system.

The design of the buffer system includes modelling the structure and the behavior of the buffer and its environment using SARA's SL1 (Structural Language 1) and GMB (Graph Model of Behavior) tools. The requirements of the buffer are checked by simulating the behavior of the buffer using the GMB Simulator. Formal analysis of the GMB models is used to detect potential deadlocks caused by synchronization mechanisms.

As an example of composition, two instances of the buffer building-block are used in the design of a larger system. Structural and behavioral models are defined, linked, and then simulated to determine if requirements are met.

Attendees will be able to enter their own models and to exercise all of the SARA tools. A transcript for each session will be automatically generated by SARA and mailed to interested parties at a later date.

4. SARA Literature:

[DroJ80] Drobman, J. "Building Block Modeling Methodology for Composition of Microprocessors-Based Digital Systems," Ph.D. Dissertation, Computer Science Department, University of California, Los Angeles, July 1980.

[EstG78] Estrin, G. "A Methodology for design of digital systems - supported by SARA at the age of one," AFIPS, Proceedings of the National Computer Conference, June 1978.

[FenR80] Fenchel, R.S "Interactive Systems with Integral Help," Ph.D. Dissertation, Computer Science Department, University of California, Los Angeles, July 1980.

[PenM80] Penedo, M.H. "The Use of a Module Interface Description in the Synthesis of Reliable Software Systems," Ph.D. Dissertation, Computer Science Department, University of California, Los Angeles, November 1980.

[PenM81] Penedo, M.H., et. al. "An Algorithm to support Code-Skeleton Generation for Concurrent Systems," Proceedings of the 5th International Conference on Software Engineering, San Diego, California, March 1981.

[RazR80a] Razouk, R. "Computer-Aided Design and Evaluation of Digital Computer Systems," Ph.D. Dissertation, Computer Science Department, University of California, Los Angeles, August 1980.

[RazR80b] Razouk, R. and G. Estrin "Modeling and Verification of Communication Protocols in SARA: The X.21 Interface," IEEE Transactions on Computers, December 1980.

[SamA81] Sampaio, A.B.C. "A Scheme of Attributes for Checking Design Inconsistencies," Ph. D. Dissertation, Computer Science Department, University of California, Los Angeles, to be completed 1981.

[WinJ80] Winchester, J. "Requirements Definition and its Interface to the SARA Design Methodology for Computer-Based Systems," Ph.D. Dissertation, Computer Science Department, University of California, Los Angeles, November 1980.

5. Station, Day and Time:

Station 3, Wednesday March 10, from 11:00 a.m. until 4:00 p.m.

6. The Demonstrator: Dr. Maria Heloisa Penedo

Dr. Penedo is currently employed in the software analysis and evaluation department of TRW's defense and space systems group. Her current research focuses on the design, evaluation and integration of tools to aid in software development. She received her Ph.D. in computer science from UCLA where she explored the SARA methodology as used to model and implement software systems. She also holds an M.S. in computer science from Pontificia Universidade Catolica and a B.S. in Mathematics from Universidade Santa Ursula in Brasil.

SAMPLE OF A SARA SESSION

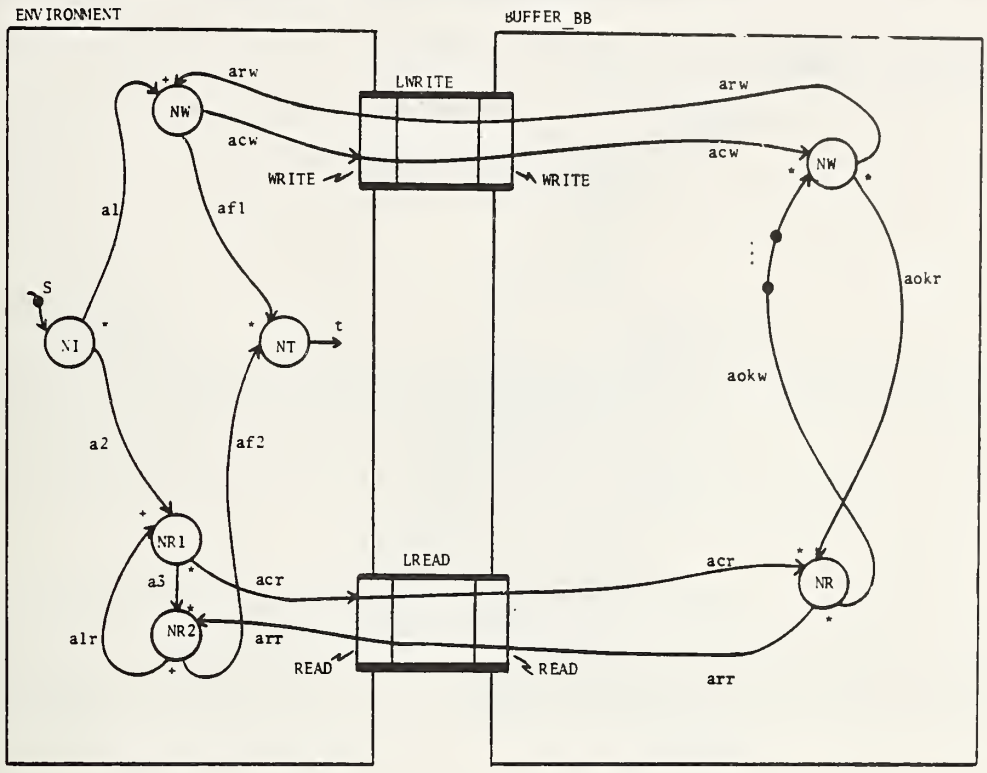
```
ec >udd>SARA>SARA_system>ec>sara
SARA Selector February 9, 1980
New or modified news:
no news changed
>/* Describe terminal characteristics to SARA.
>&terminal nl 16 ls 64
Terminal profile changed
>
>/* The following tools will be used in this demo:
>/* Structure - to process the structural model.
>/* GMB - to process the behavioral model:
>/* GMB.Translator, to process the control
>/* and data graphs and/or the mapping between
>/* the structure and behavior;  GMB.Linker,
>/* to link gmbs; GMB.PLIP - to process the
>/* interpretation for the datagraph;
>/*  GMB.Simulator, to simulate the gmb model.
>/* All tools are interactive, i.e., lines are
>/* read and processed.  Processing a model means
>/* translating it into an internal structure.
>
>/* Now we will define a working library.
>&library >udd>SARA>sl>buffer
Working library now >udd>SARA>sl>buffer
>
>/* ----------SL1 DEFINITION----------
>@structure
SARA.Structure
SL1 Translator  November 28, 1979
>
>/* The source code for the Buffer example
>/* exists in the file buffer.sll. The input
>/* will be read in and echoed.
>&input buffer.sll -echo
Input from source >udd>SARA>sl>buffer>buffer.sll started

   /* sll definition for BUFFER example */
   universe(environment, buffer_bb);
   environment < $write, $read >;
   buffer_bb < $write, $read >;
   universe(lwrite : environment$write - buffer_bb$write);
   universe(lread : environment$read - buffer_bb$read);
>
>/* The above definition specifies a module
>/* UNIVERSE with two submodules (see fig.).
>/* Next we store the current model.
>@store buffer
model stored under "buffer".
>@end;    /* ends the structure processor */
End of SL1 Translator
SARA
>
>/* ------------GMB DEFINITION------------
>@behavior;@gmb;@translator
SARA.Behavior
SARA.Behavior.GMB
SARA.Behavior.GMB.Translator
GMB Translator  Jan 16, 1980
>/* Now we define gmbs (behavioral models)
>/* to be associated with the structures.
>
@load_sll_model buffer     /* load sll_model model */
model loaded from "buffer"
  creation date: 810326

@system environment;
   /* gmb description of module environment */
   @control_graph;
      @nodes ni, nw, nr1, nr2, nt;
      @arcs s(1), al, a2, t,
            acw, arw, afl,
            acr, arr, a3, alr, af2;
         ni( s : al*a2 );
         nw( al+arw : acw+afl );
         nr1( a2+alr : a3*acr );
         nr2( a3*arr : af2+alr );
         nt( afl*af2 : t);

         acw( nw : $write );
         arw( $write : nw );
         acr( nr1 : $read );
         arr( $read : nr2 );
   @end;

   @data_graph;
      @processors pw(nw), pr2(nr2), pt(nt);
      @datasets mes_in, mes_out, input, output,
                ptr_in, ptr_out;
      @arcs di, din, do, dout, dinput, doutput,
            dpwin, dpwout, dprin, dprout, dt1, dt2;

      dinput( input : pw );
      di( pw : mes_in);
      dt1( input : pt );
      dt2( output : pt );
      doutput( pr2 : output );
      do( mes_out : pr2 );
      din( mes_in : $write);
      dout( $read : mes_out);
      dpwin( ptr_in : pw );
      dpwout( pw : ptr_in );
      dprin( ptr_out : pr2 );
      dprout( pr2 : ptr_out );
   @end;

@endsys;            /* store mapped gmb for Environment */
model stored under model name environment
model stored under "buffer".

@system buffer_bb;

.............................

>@end;   /* ends the gmb processor
percentage of gmbplx tables used = 32.0%
End of GMB Translator
SARA.Behavior.GMB
>
>/* ----------LINKING----------
>@linker buffer
SARA.Behavior.GMB.Linker
GMB Linker  November 27, 1979
model loaded from "buffer"
  creation date: 810326
Current SL1 model: buffer
>/* The gmb models were defined separately
>/* for each module. Now we compose
>/* module UNIVERSE, i.e., link the gmb
>/* models defined for its submodules.
>@compose universe
universe successfully linked
>
>/* Next we store the linked gmbs.
>@store_sll buffer
model stored under "buffer".
Current SL1 model: buffer
>@store_gmb buffer
Current GMB model: buffer
>@end;   /* end the Linker
End of GMB Linker
SARA.Behavior.GMB
>
```

```
>/* ----------PLIP DEFINITION----------
>@plip buffer
SARA.Behavior.GMB.PLIP
GMB PL1 Preprocessor  February 21, 1980
model loaded from "buffer"
  creation date: 810326
Current model:  buffer
>
>/* Now we define interpretations for
>/* the data graph using PLIP, a PL1
>/* preprocessor: procedures are associated
>/* with data processors and types are
>/* associated with datasets/dataarcs.
@system buffer_bb;

    Current SL1 system:  universe.buffer_bb
       /* plip for module buffer_bb */

       @template(din, dout)
                tmessage char(80) var;
       @template(dw, dw_in,  dr)
                tbuffer(8) char(80) var;
       @template(dprin, dpwin, dpwout, dprout)
                t1 fixed bin(15);
       @dataset (ptr_w ptr_r) @like t1 @initial(0);

       @processor pw;

          @read message @from din;
          @read ptr_w @from dpwin;
          ptr_w = mod( ptr_w, 8) + 1;
          /* we use a read-modify-write for arrays */
          @read buffer @from dw_in;
          buffer(ptr_w) = message;
          @write buffer @to dw;
          @write ptr_w @to dpwout;
       @endprocessor;

  ..........................
>/* Now we store the current plip model
>/* and tell SARA to compile the code.
>@store
***   0 errors
***   0 warnings
Do you want to compile the PLIP output?>yes
PL/I compilation in progress
PL/I 26a
Current model:  buffer
>
>@end;   /* terminate PLIP
End of GMB PL1 Preprocessor
SARA.Behavior.GMB
>
```

```
>/* ----------SIMULATOR----------
>@sim buffer
SARA.Behavior.GMB.Simulator
GMB Simulator  November 28, 1979
model loaded from "buffer"
  creation date: 810326
>
>/* The buffer building block model may now
>/* be simulated. We will test whether the
>/* buffer behaves according to its reqs.
>/* The Environment is used to generate
>/* messages (at 2 ms intervals) to be
>/* deposited in the buffer.  It also reads
>/* in the messages from the buffer (at 5 ms
>/* intervals). The writing and reading are
>/* done concurrently. Upon completion,
>/* all messages are deposited and
>/* received by the Environment, processor
>/* PT checks the output messages
>/* and prints out the result of
>/* the comparison.  If all output messages
>/* are the same as the input ones, it prints
>/* out: 'Messages were delivered correctly';
>/* otherwise it prints: 'Messages were not
>/* delivered correctly'.
>
>/* Start the simulation
>@start
messages were delivered correctly
End of simulation, time =  152000062 ns
proper termination of control graph
>
>/* Simulation ended; buffer system behaved as
>/* expected.
>@end;  /* terminate Simulator
End of GMB Simulator
SARA.Behavior.GMB
>@end
SARA.Behavior
>@end
SARA
>@end
End of SARA Selector
```

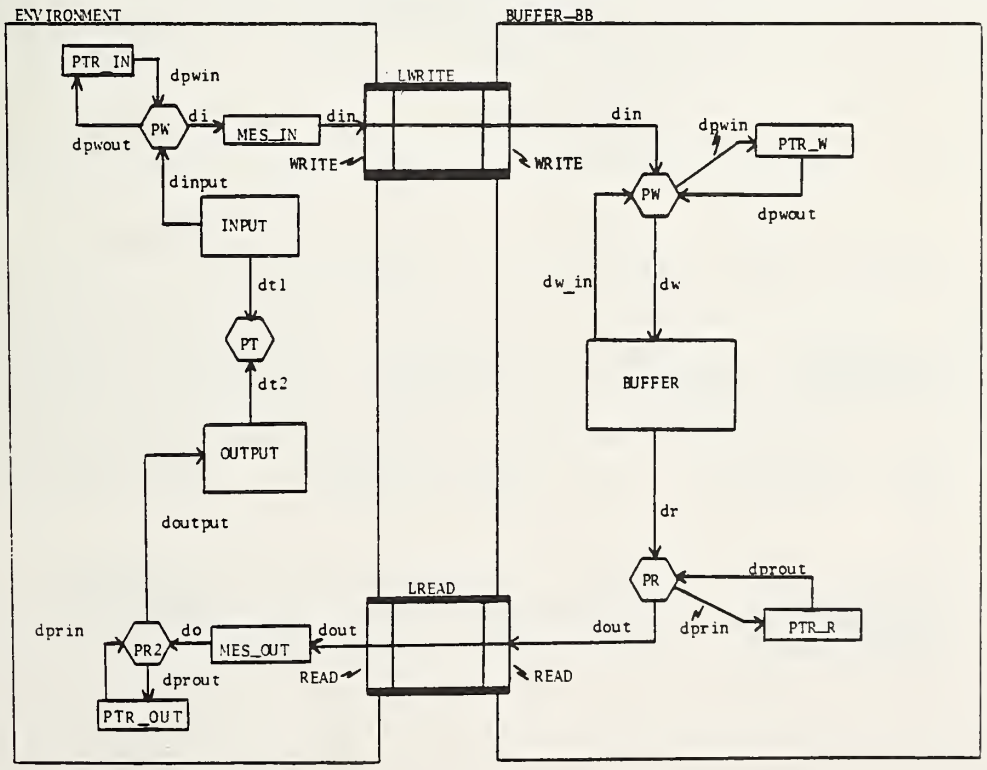
UNIVERSE

GMB CONTROL GRAPH



UNIVERSE

GMB DATA GRAPH



FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . VHLL INPUT
. . . SL1
. . . GMB
. . . BNF
. . TEXT INPUT
FUNCTION
. TRANSFORMATION
. . TRANSLATION
. . FORMATTING
. . RESTRUCTURING
. STATIC ANALYSIS
. . DATA FLOW ANALYSIS
. . STRUCTURE CHECKING
. . CROSS REFERENCE
. . CONSISTENCY CHECKING
. . COMPLETENESS CHECKING
. . SCANNING
. DYNAMIC ANALYSIS
. . SIMULATION
OUTPUT
. USER OUTPUT
. . GRAPHICS
. . LISTINGS
. . USER-ORIENTED TEXT
. . . DOCUMENTATION
. . . ON-LINE ASSISTANCE
. MACHINE OUTPUT
. . SOURCE CODE OUTPUT
. . DATA OUTPUT
. . PROMPTS

IMPLEMENTATION LANGUAGE: PL/1, TOOL PORTABLE: NO, TOOL SIZE: 25000 LINES OF PL/1 SOURCE

COMPUTER (OTHER HARDWARE): HONEYWELL, OS (OTHER SOFTWARE): MULTICS

TOOL AVAILABLE: YES, PUBLIC DOMAIN: YES, TOOL SUPPORTED: NO

CONTACT: G. ESTRIN, UNIVERSITY OF CALIFORNIA, COMPUTER SCIENCE DEP, BOELTER HALL 3732, LOS ANGELES, CA, 90024, USA, 213-825-8878

ARGUS in the Microcomputer Environment

William C. King

Boeing Computer Services Company

Seattle, Washington

1. Introduction

ARGUS is an integrated collection of software management, design, testing and maintenance tools. ARGUS is being developed by the Space and Military Applications Division of Boeing Computer Services Company.

This proposal describes the demonstration of ARGUS on a microcomputer.

2. Summary of ARGUS in the Microcomputer Environment

ARGUS on a microcomputer provides a number of capabilities similar to those available with ARGUS on main-frame computers, but also provides some important new capabilities affordable with the dedicated processing power and high display band width of a microcomputer.

Through a menu-driven interface, ARGUS provides access to Pascal and Fortran 77 compilers, link editor, DAPPER -- A Dynamic Analyzer for Pascal, Pascal source cross reference generator, and VED -- a powerful text and graphics editor. VED has been tailored for the creation of "data flow" diagrams and viewfoils, in addition to many other graphical charts for arbitrary documentation purposes. Complementing these capabilities is a set of compatable print, plot and file-manipulation utilities.

Plans for further development include linking the excellent user-interface of the micro-based version of ARGUS to the main-frame version of ARGUS.

3. A Scenario for Demonstration

The demonstration of ARGUS will consist of a "canned" demonstration of VED -- the text/graphics editor, and a walk-through of the ARGUS menu interface. Tools Fair participants will be offered the ability to develop their own graphic creations by hands-on use of the VED program. Samples of ARGUS produced graphics (including "data flow" diagrams) will be available for the attendees.

4. ARGUS Literature

Available within the Boeing Company:

The ARGUS Microcomputer Environment (Reference Manual), Bill King, February 10, 1981

DAPPER-- Dynamic Analyzer: Pascal Program Execution reporter (User's Manual), John Joseph Chilenski

Available outside Boeing:

"Concepts and Prototypes of ARGUS -- A Progress Report on the ARGUS Project," Leon G. Stucki and Harry D. Walker (Contributed chapter to Software Engineering Environments, Edited by Horst Huenke, North-Holland Publishing Company -Amsterdam - New York - Oxford 1981)

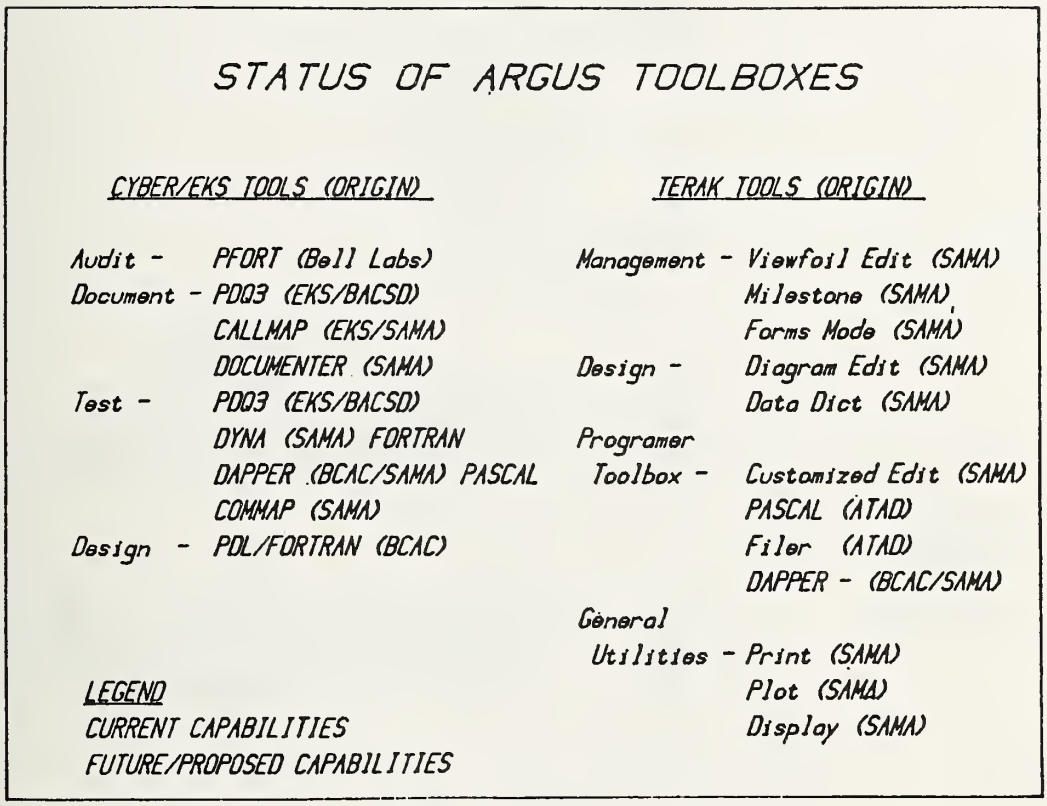
5. Station, Day, and Times

ARGUS will be demonstrated at Station 4, Wednesday March 11 throughout the day. A special extended presentation will be offered on Thursday morning in the special presentation room.

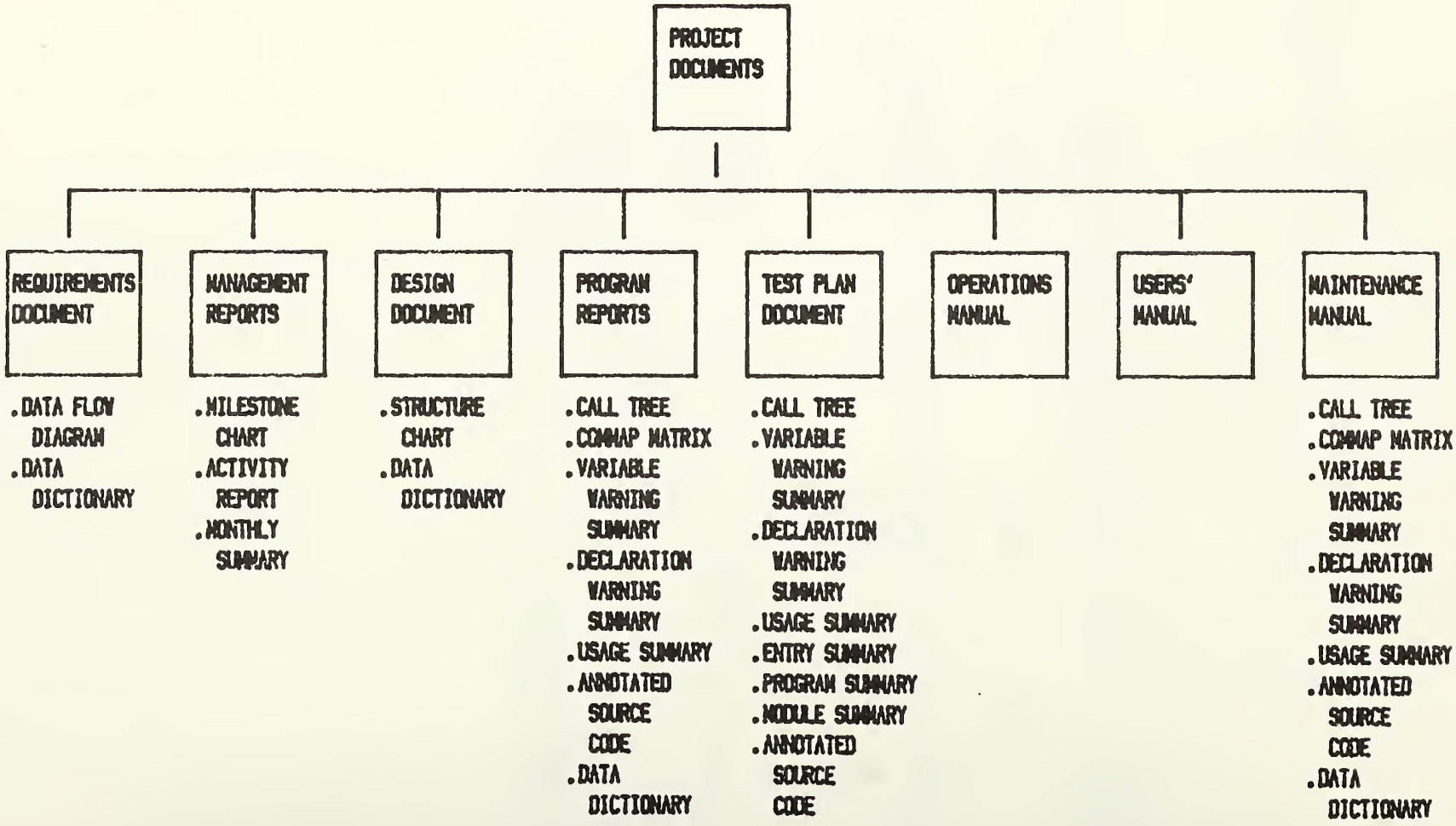
6. The Demonstrator

William C. King

Since joining the Software Engineering Technology Group within the Space and Military Applications Division of Boeing Computer Services Company in 1980, Mr. King has participated in several software engineering research projects. Most notably, he has been responsible for implementing the ARGUS environment on microcomputer-based systems. Prior to joining Boeing Computer Sevices Company, William C. King was a member of the scientific staff at Bell Northern Research, Inc., Palo Alto, with the Advanced Business Systems Group. Mr. King received his undergraduate degree in computer science from Washington State University, where he was well-known as a co-author of "bg" a backgammon-playing program.


STATUS OF ARGUS TOOLBOXES
CYBER/EKS TOOLS (ORIGIN)
Audit - PFORT (Bell Labs)
Document - PDQ3 (EKS/BACSD)
CALLMAP (EKS/SAMA)
DOCUMENTER (SAMA)
Test - PDQ3 (EKS/BACSD)
DYNA (SAMA) FORTRAN
DAPPER (BCAC/SAMA) PASCAL
COMMAP (SAMA)
Design - PDL/FORTRAN (BCAC)
LEGEND
CURRENT CAPABILITIES
FUTURE/PROPOSED CAPABILITIES
TERAK TOOLS (ORIGIN)
Management - Viewfoil Edit (SAMA)
Milestone (SAMA)
Forms Mode (SAMA)
Design - Diagram Edit (SAMA)
Data Dict (SAMA)
Programer Toolbox - Customized Edit (SAMA)
PASCAL (ATAD)
Filer (ATAD)
DAPPER - (BCAC/SAMA)
General Utilities - Print (SAMA)
Plot (SAMA)
Display (SAMA)

DOCUMENTATION TREE



FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . TEXT INPUT
. . CODE INPUT
. . . FORTRAN
. . . PASCAL
. . . FORTRAN 77
FUNCTION
. TRANSFORMATION
. . TRANSLATION
. . EDITING
. . INSTRUMENTATION
. STATIC ANALYSIS
. . MANAGEMENT
. . . FILES MANAGEMENT
. DYNAMIC ANALYSIS
. . COVERAGE ANALYSIS
OUTPUT
. USER OUTPUT
. . DIAGNOSTICS
. . USER-ORIENTED TEXT
. . GRAPHICS
. . TABLES
. . LISTINGS

IMPLEMENTATION LANGUAGE: PASCAL

TOOL PORTABLE: YES

TOOL AVAILABLE: NO, PUBLIC DOMAIN: NO

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): UNDER PROTOTYPE DEVELOPMENT, CURRENTLY FOR INTERNAL BOEING USE

TOOL SUPPORTED: YES, TOOL SUPPORT: BOEING COMPUTER SERVICES COMPANY

CONTACT: LEON G. STUCKI, BOEING COMPUTER SERVICES COMPANY, P.O. BOX 24346 M/S 9C-Ø3, SEATTLE, WA, 98124, USA, 2Ø6-575-5118
WILLIAM C. KING, BOEING COMPUTER SERVICES, PO BOX 24346, SEATTLE, WA, 98124,

DYNA: A Tool From the ARGUS Toolbox

Leon G. Stucki

Boeing Computer Services Company

Seattle, Washington

1. Introduction

This proposal is in response to the call for software engineering developers to demonstrate their accomplishments at the 5th International Conference on Software Engineering in San Diego, March 9 - 12, 1981.

This proposal describes the demonstration of DYNA. DYNA is a dynamic analyzer for FORTRAN programs which is part of the ARGUS toolbox being developed by the Space and Military Applications Division of Boeing Computer Services.

2. Summary of DYNA

DYNA is a tool for FORTRAN Programs which allows the user to see the dynamic behavior of a module while it is executing test data. The test data used is that of the users or developers. No modifications to either the program or its test data are required. DYNA operates in three steps. During the preprocessing step, probes (Additional FORTRAN statements) are automatically inserted in the source code and the "instrumented source" code is compiled. During the execution step, counts are made of the number of times each statement is executed by the test data. The counts may be accumulated with the counts from previous executions, if desired. During the post processing step, the execution data is formatted into reports.

The reports include an Entry Summary, a Program Summary, Module Summaries and an Annotated Source Listing. The Entry Summary documents the number of times each module was called while executing the test data. The Program Summary and Module Summaries categorize the number of different types of statements in the program or module and the percentage of each which were executed. This information provides a measure of how thoroughly the program has been tested. The Annotated Source Listing provides detailed information about the number of times each statement or branch was executed.

3. A Scenario for a DYNA Demonstration

The demonstration of DYNA will consist of two parts. The first part will be a description of the reports produced by DYNA. The second part of the demonstration will show how DYNA is accessed through the ARGUS interface.

4. DYNA Literature

Requirements for ARGUS An Advanced Software Engineering Workbench (Concept Definition) A discussion of the concepts, objectives and dvelopment philosophy of the ARGUS project. (Available within Boeing.)

DYNA User's Manual EKS Version
A user's manual for the dynamic analyzer for FORTRAN program.
(Available within Boeing.)

"Concepts and Prototypes of ARGUS -- A Progress Report on the ARGUS Project," Leon G. Stucki and Harry D. Walker (Contributed chapter to Software Engineering Environments, Edited by Horst Huenke, North-Holland Publishing Company -Amsterdam - New York - Oxford 1981)

5. Station, Day, and Time

DYNA on EKS will be demonstrated at Station 4,Wednesday March 11 throughout the day. A special extended presentation will be offered on Thursday morning in the special presentation room.

6. The Demonstrator

Leon Stucki

Dr. Leon G. Stucki is Manager of Software Engineering Technology for the Space and Military Applications Division of BCS. He is responsible for coordinating technology development and dissemination throughout the entire division. He also serves as liason between the division and other corporate organizations involved with advanced technology. He is currently responsible for an internal productivity research and development project ARGUS aimed at creating an advanced computer aided software engineering environment. Other current activities within the SAMA Division include support to contract research in the following areas of software engineering: software management technology, requirements analysis/verification, design analysis/verification, and automated verification of code. Prior to his career with BCS, Dr. Stucki spent 9 years with McDonnell Douglas Astronautics Company. There he had broad experience in the design, implementation, testing, validation and verification of advanced military, space, and commercial software systems. He received his Ph.D., Computer Science from UCLA, 1976; M.B.A., Business Administration, University of California, 1970; and B.A., Mathematics, University of Utah, 1968. Dr. Stucki is a member of the ACM, and also currently serves on the Executive Board of the IEEE Technical Committee on Software Engineering. He is a member of the editorial board for the widely circulated IEEE Transactions on Software Engineering. He is also serving on the editorial board of The Journal of Systems and Software. Dr. Stucki is Program Chairman of the 5th International Conference of Software Engineering. He was Program Co-Chairman for the 4th International Software Engineering Conference held in Munich, Germany in 1979, has served on various other program committees, and was Chairman of the 1975 NBS-ACM-IEEE Workshop on Currently Available Program Testing Tools.

VER 1.1 DYNA PROGRAM SUMMARY

LAST RUN DATE: 81/03/05. LIST DATE: 81/03/05.
TIME: 11.15.21. TIME: 11.17.00.

SUMMARY OF 2 RUNS PROGRAM SOLVE

CLASSIFICATION OF STATEMENTS

STATEMENT TYPE	NUMBER PRESENT	PERCENT OF TOTAL	NUMBER MONITORED	PERCENT MONITORED
COMMENT	12	20		
DECLARATIVE	12	20		
EXECUTABLE	37	61	37	100
UNRECOGNIZED	0	0		
TOTAL	61			

EXECUTABLE STATEMENTS

STATEMENT TYPE	TOTAL STATEMENTS PRESENT	MONITORED STATEMENTS EXECUTED	% EXECUTED
ASSIGNMENT	11	9	82
DO	1	1	100
TRANSFER	19	11	85
SIMPLE GO TO	7	5	71
COMPUTED GO TO	1	1	100
BRANCHES	4	3	75
OUT-OF-RANGE		0	0
ASSIGN GO TO	0		
LOGICAL IF	2	2	100
TRUE	2	1	50
FALSE	2	2	100
ARITHMETIC IF (3 BR	1	1	100
NEGATIVE	1	1	100
ZERO	1	0	0
POSITIVE	1	1	100
ARITHMETIC IF (2 BR)	0		
ZERO	0		
NON-ZERO	0		
CALL	2	2	100
RETURNS	2	2	100
INPUT/OUTPUT	7	5	71
OTHER	5	4	80

SUMMARY TOTALS

MODULES	3	3	100
EXECUTABLE	37	30	81
BRANCHES	20	15	75

LAST RUN DATE: 81/03/05. LIST DATE: 81/03/05.
TIME: 11.15.21. TIME: 11.17.00.
SUMMARY OF 2 RUNS PROGRAM SOLVE
PAGE 1

---

3 OUT OF 3 MODULES ENTERED
100 % MODULES ENTERED

MODULES ENTERED

MODULE NAME	TIMES ENTERED
PROGRAM SOLVE	2
SUBROUTINE LINEAR	2
SUBROUTINE DISCRM	3

VER 1.1 DYNA MODULE SUMMARY

LAST RUN DATE: 81/03/25. LIST DATE: 81/03/25.
TIME: 11.15.21. TIME: 11.17.00.

SUMMARY OF 2 RUNS SUBROUTINE LINEAR

CLASSIFICATION OF STATEMENTS

STATEMENT TYPE	NUMBER PRESENT	PERCENT OF TOTAL	NUMBER MONITORED	PERCENT MONITORED
COMMENT	0	0		
DECLARATIVE	2	33		
EXECUTABLE	4	67	4	100
UNRECOGNIZED	0	0		
TOTAL	6			

EXECUTABLE STATEMENTS

STATEMENT TYPE	TOTAL STATEMENTS PRESENT	MONITORED STATEMENTS EXECUTED	% EXECUTED
ASSIGNMENT	1	1	100
DO	0		
TRANSFER	1	1	100
SIMPLE GO TO	0		
COMPUTED GO TO	0		
BRANCHES	0		
OUT-OF-RANGE			
ASSIGN GO TO	0		
LOGICAL IF	1	1	100
TRUE	1	0	0
FALSE	1	1	100
ARITHMETIC IF (3 BR)	0		
NEGATIVE	0		
ZERO	0		
POSITIVE	0		
ARITHMETIC IF (2 BR)	0		
ZERO	0		
NON-ZERO	0		
CALL	0		
RETURNS	0		
INPUT/OUTPUT	0		
OTHER	2	1	50

SUMMARY TOTALS

EXECUTABLE	4	3	75
BRANCHES	2	1	50

TOTAL TIMES ENTERED: 2

DYNA EXECUTION SUMMARY

RUN DATE	RUN TIME	RUN DESCRIPTION
81/03/05	11.08.38	INPUT FILE: IN1
81/03/05	11.15.21	INPUT FILE: IN2

DYNA ANNOTATED SOURCE LISTING

LAST RUN DATE: 81/03/05 LIST DATE 81/03/05.
LAST RUN TIME: 11.15.21 LIST TIME 11.17.00.
SUMMARY OF 2 RUNS

```
LINE#                    SOURCE                          MONITOR COUNTS
                                                        TOTAL   CONDITIONAL

    1       PROGRAM SOLVE(INFIL,OUTFIL,T...          2
    2       INTEGER COUNT,JUMP
    3  001  FORMAT( I10 )
    4  002  FORMAT(3F10.2)
    5  051  FORMAT(1X,F10.2," X**2 +",F1...
    6     *         /,5X,"X1 = ",F10.2,  "...
    7     *         /,5X,"X2 = ",F10.2,  "...
    8  052  FORMAT(1X,F10.2," X**2 +",F1...
    9     *         /,5X,"X1 = ",F10.2,/,...
   10 053   FORMAT(1X,F10.2," = 0",/,5X,...
   11 054   FORMAT(1X,F10.2," = 0",/,5X,...
   12       READ(7,001)COUNT                         2
   13C
   14       DO 100 I=1,COUNT                         2   EXECUTED   :      8
   15          READ(7,002)A,B,C                      8
   16          IF (A .NE. 0.) GOTO 10                8   TRUE       :      3
                                                         FALSE      :      3
   17             CALL LINEAR(B,C,X),RET...          2   RETURNED   :  **0**
   18    5        JUMP = 4                           2
   19             GOTO 50                            2
   20C
   21   10     CALL DISCRM(A,B,C,D)                  3   RETURNED   :      3
   22          IF (D) 20,30,40                       3   NEGATIVE   :      2
                                                         ZERO       :  **0**
                                                         POSITIVE   :      1
   23C
   24   20        REALX = -B/(2*A)                   2
   25             COMPX = SQRT(-D)/(2*A)...          2
   26             JUMP = 1                           2
   27             GOTO 50                            2
   28C
   29   30        REALX  = -B/(2*A)                **0**
   30             JUMP = 2                         **0**
   31             GOTO 50                          **0**
   32C
```

DYNA ANNOTATED SOURCE LISTING

LAST RUN DATE: 81/03/05 LIST DATE 81/03/05.
LAST RUN TIME: 11.15.21 LIST TIME 11.17.00.
SUMMARY OF 2 RUNS

```
LINE#                          SOURCE                          MONITOR COUNTS
                                                              TOTAL   CONDITIONAL
   33   40       REAL1 = (-B + SQRT(D))...                      1
   34            REAL2 = (-B - SQRT(D))...                      1
   35            JUMP =3                                        1
   36C
   37   50    GOTO(80,70,80,90), JUMP                           5   LABEL     80:      2
                                                                    LABEL     70:   **0**
                                                                    LABEL     80:      1
                                                                    LABEL     90:      2
                                                                    OUT OF RANGE:   **0**
   38C
   39   80    WRITE(8,951) A,B,C,REAL8,...                      2
   40         GOTO 100                                          2
   41C
   42   70    WRITE(8,952)A,B,C,REALX,R...                   **0**
   43         GOTO 100                                       **0**
   44C
   45   80    WRITE(8,952) A,B,C,REAL1,...                      1
   46         GOTO 100                                          1
   47C
   48   90    WRITE(8,953)B,C,X                                 2
   49         GOTO 100                                          2
   50C
   51   99    WRITE(8,954)C                                  **0**
   52C
   53  100 CONTINUE                                           5
   54       STOP                                              1
   55       END
   56       SUBROUTINE LINEAR (B,C,X),RE...                   2
   57       REAL B,C,X
   58       IF (B .EQ. 0)   GO TO 10                          2   TRUE      :   **0**
                                                                    FALSE     :      2
   59          X = -C/B                                       2
   60          RETURN M                                       2
   61 10    RETURN N                                       **0**
   62       END
   63       SUBROUTINE DISCRM (A,B,C,D)                       3
   64       REAL A,B,C,D
   65       D = B*B - 4*A*C                                   3
   66       RETURN                                            3
   67       END
```

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . CODE INPUT
. . . FORTRAN
. . . FORTRAN 66
FUNCTION
. TRANSFORMATION
. . INSTRUMENTATION
. DYNAMIC ANALYSIS
. . COVERAGE ANALYSIS
OUTPUT
. USER OUTPUT
. . TABLES
. . LISTINGS
. MACHINE OUTPUT
. . SOURCE CODE OUTPUT
. . . FORTRAN
. . . FORTRAN 66

IMPLEMENTATION LANGUAGE: FORTRAN 77

TOOL PORTABLE: YES

TOOL AVAILABLE: NO, PUBLIC DOMAIN: NO

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): FOR INTERNAL BOEING USE

TOOL SUPPORTED: YES, TOOL SUPPORT: BOEING COMPUTER SERVICES COMPANY

CONTACT: LEON G. STUCKI, BOEING COMPUTER SERVICES COMPANY, P.O. BOX 24346 M/S 9C-Ø3, SEATTLE, WA, 98124, USA, 2Ø6-575-5118

COMMAP: A Tool from the ARGUS Toolbox

Leon G. Stucki

Boeing Computer Services Company

Seattle, Washington

1. Introduction

This proposal is in response to the call for software engineering developers to demonstrate their accomplishments at the 5th International Conference on Software Engineering in San Diego, March 9 - 12, 1981.

This proposal describes the demonstration of COMMAP. COMMAP is a common block analysis tool which is part of the ARGUS toolbox being developed by the Space and Military Applications Division of Boeing Computer Services.

2. Summary of COMMAP

COMMAP is a static analyzer for FORTRAN programs. Operating on existing source code, it produces a matrix cross-referencing variables in common blocks versus the subroutines that use them. The matrix specifies whether a variable is referenced or defined within a subroutine. It also analyzes the information in the matrix and reports on potential errors in the use of the variable (for example, variables which are referenced, but never defined).

3. A Scenario for COMMAP Demonstration

The demonstration of COMMAP will consist of two parts. The first part will be a description of the matrices and reports produced by COMMAP. The second part of the demonstration will show how COMMAP is accessed through the ARGUS interface.

4. ARGUS Literature

The ARGUS reference library includes the following documents.

Requirements for ARGUS An Advanced Software Engineering Workbench (Concept Definition)
A discussion of the concepts, objectives and development philosophy of the ARGUS project. (Available within Boeing.)

"Concepts and Prototypes of ARGUS -- A Progress Report on the ARGUS Project," Leon G. Stucki and Harry D. Walker (Contributed chapter to Software Engineering Environments, Edited by Horst Huenke, North-Holland Publishing Company -Amsterdam - New York - Oxford 1981)

5. Station, Day, and Time

The ARGUS Toolbox on CYBER/EKS will be demonstrated at Station 4,Wednesday March 11 throughout the day. A special extended presentation will be offered on Thursday morning in the special presentation room.

6. The Demonstrator

Leon Stucki

Dr. Leon G. Stucki is Manager of Software Engineering Technology for the Space and Military Applications Division of BCS. He is responsible for coordinating technology development and dissemination throughout the entire division. He also serves as liason between the division and other corporate organizations involved with advanced technology. He is currently responsible for an internal productivity research and development project ARGUS aimed at creating an advanced computer-aided software engineering environment. Other current activities within the SAMA Division include support to contract research in the following areas of software engineering: software management technology, requirements analysis/verification, design analysis/verification, and automated verification of code. Prior to his career with BCS, Dr. Stucki spent 9 years with McDonnell Douglas Astronautics Company. There he had broad experience in the design, implementation, testing, validation and verification of advanced military, space, and commercial software systems. He received his Ph.D., Computer Science from UCLA, 1976; M.B.A., Business Administration, University of California, 1970; and B.A., Mathematics, University of Utah, 1968. Dr. Stucki is a member of the ACM, and also currently serves on the Executive Board of the IEEE Technical Committee on Software Engineering. He is a member of the editorial board for the widely circulated IEEE Transactions on Software Engineering. He is also serving on the editorial board of The Journal of Systems and Software. Dr. Stucki is Program Chairman of the 5th International Conference of Software Engineering. He was Program Co-Chairman for the 4th International Software Engineering Conference held in Munich, Germany in 1979, has served on various other program committees, and was Chairman of the 1975 NBS-ACM-IEEE Workshop on Currently Available Program Testing Tools.

COMMON BLOCK MATRIX

```
                                   COMMON BLOCK MATRIX
      STRIP  1
      KEY :  R - ONLY REFERENCED
             D - ONLY DEFINED
             B - BOTH REFERENCED AND DEFINED
             ? - UNKNOWN (USED IN A CALL ETC.)
             * - COMMON BLOCK DEFINED
            ** - WARNING EQUIVALENCED VARIABLE OR COMMON MULTIPLY DEFINED


               :S I O:S D L:V D
               :E N U:O I I:E I
               :T P T:L S N:L S
               :U U P:V C E:
               :P T U:E   A:
               :    T:    R:
               :     :     :
-------------------------------
**FIREWK       :     :* *  :
    HEIGHT     :     :     :
    XANGLE     :     :     :
    XSPEED     :     :     :
    YANGLE     :     :     :
    YSPEED     :     :     :
               :     :     :
  PHYCON       :    *:     :
    G          :     :     :
    PI         :    R:     :
               :     :     :
**QUAD         :*   *:* *  :
    A          :     :  D  :
    B          :D    :     :
    C          :R    :     :
               :     :     :
**RCOMP        :* *  :* * *:* *
    TIME       :  D  :  R  :
    XDIST      :     :     :
  **XGVEL      :  D  :  D D:D B
    YDIST      :     :     :
    YGUEL      :D R  :  R  :
               :     :     :
  SPACEN       :*    :     :
    ELVSP      :     :     :
    GNG        :     :     :
    LKUNI      :     :     :
    MVEL       :D    :     :
               :     :     :
```

COMMAP VARIABLE WARNING SUMMARY

LIST DATE: 81/03/05.
PAGE 1 TIME: 09.20.26.

COMMON BLOCKS	VARIABLES REFERENCED BUT NEVER DEFINED	VARIABLES DEFINED BUT NEVER REFERENCED	VARIABLES WITH WARNINGS*	VARIABLES NEVER USED	VARIABLES USED BY ONLY ONE MODULE+
**FIREWK				HEIGHT	
				XANGLE	
				XSPEED	
				YANGLE	
				YSPEED	
PHYCON	PI			G	PI
**QUAD	C	B			C
		A			B
					A
**RCOMP			XGVEL	XDIST	
				YDIST	
SPACEN		MVEL		ELVSP	MVEL
				GNG	
				LKUNI	

** SEE DECLARATION WARNING SUMMARY
* VARIABLE EQUIVALENCED TO ANOTHER
+ NOT AN ERROR BUT WOULD REDUCE DATA COUPLING
IF ISOLATED IN A SEPARATE COMMON BLOCK(FTN4)
OR DECLARED WITH A SAVE COMMAND (FTN5)

COMMAP DECLARATION WARNING SUMMARY

LIST DATE: 81/03/05.
PAGE 1 TIME: 09.20.26.

UNNECESSARY COMMON BLOCK DECLARATIONS

COMMON BLOCKS	MODULES USING NO VARIABLES
FIREWK	SOLVE DISC
QUAD	OUTPUT SOLVE
RCOMP	SOLVE

COMMON BLOCKS THAT ARE NOT USED BY ANY MODULE

FIREWK

COMMON BLOCKS THAT ARE DECLARED DIFFERENTLY
(E.G. DIFFERENT VARIABLE NAMES, DIFFERENT LENGTHS...)

FIREWK QUAD RCOMP

COMMON USAGE SUMMARY

LIST DATE: 81/03/05.

PAGE 1 TIME: 09.20.26.

NECESSARY COMMON BLOCK DECLARATIONS

COMMON BLOCKS	MODULES USING VARIABLES
PHYCON	OUTPUT
QUAD	SETUP DISC
RCOMP	SETUP INPUT DISC LINEAR VEL DIS
SPACEN	SETUP

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . CODE INPUT
. . . FORTRAN
FUNCTION
. STATIC ANALYSIS
. . CROSS REFERENCE
. . ERROR CHECKING
OUTPUT
. USER OUTPUT
. . DIAGNOSTICS
. . TABLES

IMPLEMENTATION LANGUAGE: FORTRAN

TOOL PORTABLE: YES

TOOL AVAILABLE: NO, PUBLIC DOMAIN: NO

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): FOR INTERNAL BOEING USE

TOOL SUPPORTED: YES, TOOL SUPPORT: BOEING COMPUTER SERVICES COMPANY

CONTACT: LEON G. STUCKI, BOEING COMPUTER SERVICES COMPANY, P.O. BOX 24346 M/S 9C-03, SEATTLE, WA, 98124, USA, 206-575-5118

LOGICFLOW:
A Software Design & Analysis Tool

S. Moy, G. Nielsen, R. Sanchez, T. Wallace, S. McWethy

1. Introduction

This proposal is in response to the call for software engineering tool developers to demonstrate their accomplishments at the 5th International Conference on Software Engineering in San Diego, March 9-12, 1981.

We will be demonstrating the LOGICFLOW system developed by Logicon, Inc. The demonstration will consist of a cartridge tape driven, pre-recorded computer output of typical computer sessions.

2. Summary of LOGICFLOW

The LOGICFLOW System is a Logicon-developed software system which aids the design, development, analysis and documentation of reliable software.

During the design phase, LOGICFLOW accepts program design language (PDL) and produces a graphic representation of that design in flowchart form for evaluation. Since LOGICFLOW analyzes the syntax of the input to produce the graphic representation at this stage, it also checks for basic logic errors such as improper loop constructs. Metrics of that design can be obtained by invoking a metrics evaluator. This feature provides the analyst normalized metrics on selected design characteristics such as structuredness, simplicity, and complexity. A cross-reference of all design names can be obtained using the cross-reference generator.

The LOGICFLOW Flowcharter automatically produces high-quality flowcharts from Design Language or FORTRAN source. The flowcharts can be output to a plotter, a graphics display unit or a line printer. Unstructured code can be flowcharted as is or the user can choose to have it drawn as a functionally equivalent structured chart.[1] Once the design has been finalized to a low level,

1. Dahl, Dykstra, and Hoare, "Structured Programming", Academic Press, 1972

LOGICFLOW translates the Design Language into FORTRAN or JOVIAL source code, thereby maintaining the integrity of the design and eliminating coding errors during translation.

3. Scenario of the LOGICFLOW Demonstration

The demonstration of the LOGICFLOW system will consist of verbal descriptions and visual demonstrations of each of the LOGICFLOW features. The demonstration will illustrate the generation of a simple software design from idea-conception to translation into a higher-order-language source code.

The demonstration will take about forty minutes with time reserved for questions at the end.

4. LOGICFLOW Literature

The LOGICFLOW reference library is composed of the following documents:

- Requirements Documents
  A detailed description of the requirements for each tool developed in the LOGICFLOW System.
- Design Documents
  A detailed description of function and data flow for each procedure in each tool is given.
- Users Guide
  A comprehensive users guide to the LOGICFLOW System with a section for beginning or one-time users and another section for the sophisticated user.

5. Station, Day and Time

Station 5, Wednesday, March 11, from 10:00 a.m. until 4:00 p.m.

6. The Demonstrators

Susan Moy

Susan Moy is responsible for the Design Metric Study in the Systems Evaluation Department. The purpose of this study is to identify and implement techniques for measuring software design quality. Her work includes the identification

of quantifiable characteristics of good software design and the enhancement of a prototype design metric evaluator program. Ms. Moy's prior work includes the evaluation of the Computer-Aided Design and Specification Analysis Tool (CADSAT) and its applicability to program specifications. She has also worked on a research project on the technologies of software conversion, from assembly to higher-order language, and from one computer to another. Before that she participated in the verification and validation of the Metric Integrated Processing System library. She has a B.A. in mathematics from Hampshire College (1974) and an M.A. in statistics from the University of Rochester (1976).

Gary Nielsen

Gary Nielsen, a member of the C^3 Software Department, is task leader for the development of the MX JOVIAL/J73 Editor, a static code analyzer to be used for Logicon's MX PATE/NSCCA activities. Mr. Nielsen is also assisting the development of the JOVIAL/J73 Analyzer and Flowcharter Program, an MX PATE/NSCCA support tool that interfaces with the LOGICFLOW system. This program will execute on the VAX 11/780 and produce interface files to drive the LOGICFLOW Flowcharter. Mr. Nielsen has also been responsible for a software testing study evaluating the applicability of new testing methodologies to MX programs. Before his present assignment, Mr. Nielsen was responsible for Logicon's initial Design Metrics Study. He has a B.S. in mathematical sciences from Stanford University (1974) and an M.S. in computer science from the University of Southern California (1976).

Robert Sanchez

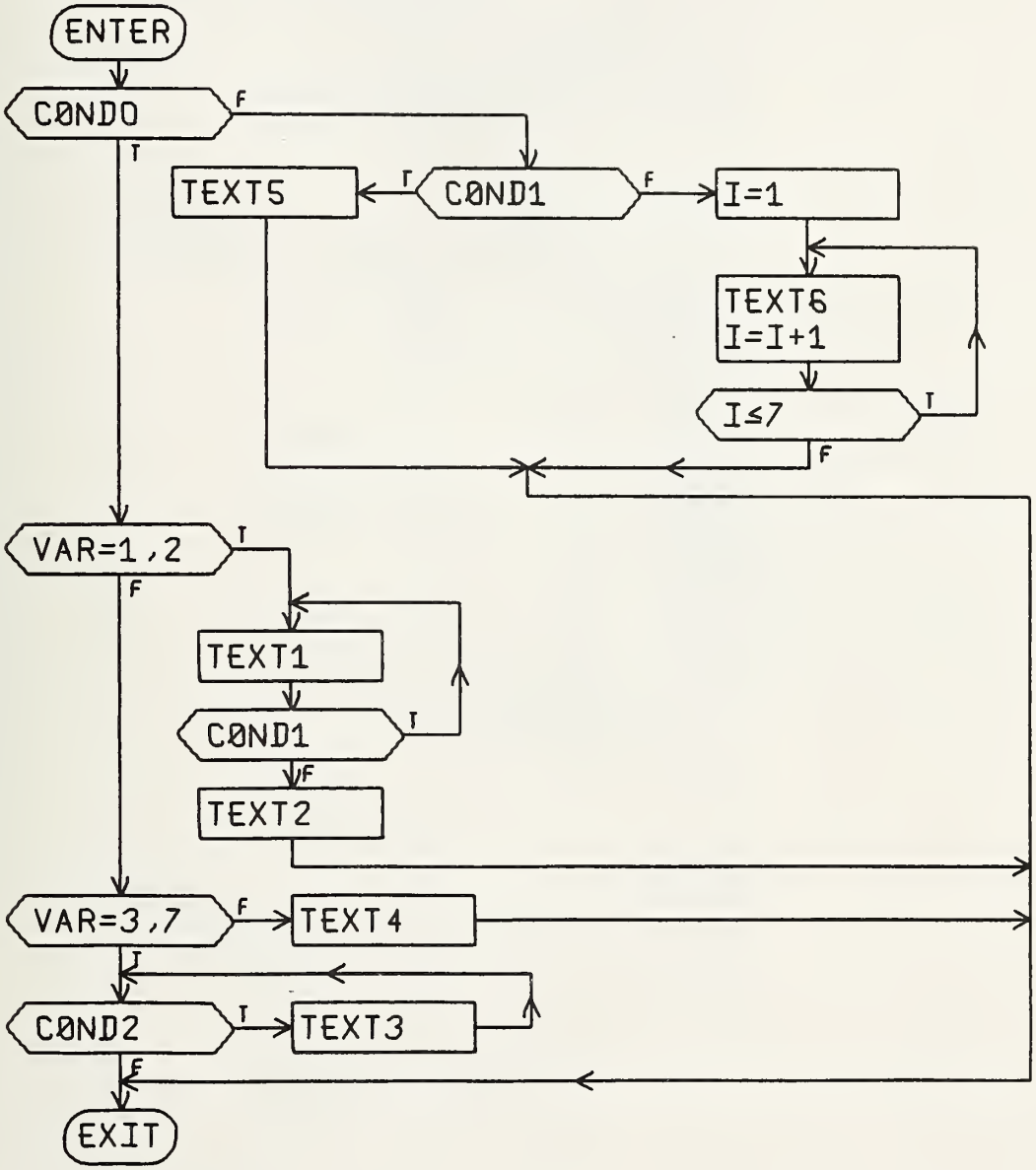
Robert Sanchez, a programmer analyst in the Systems Evaluation Department, is responsible for the implementation of the LOGICFLOW cross-reference generator feature. Earlier he performed verification and validation tasks on the Minuteman System. Mr. Sanchez has a B.A. in mathematics from California State University, Dominguez Hills (1976).

Thomas Wallace

Thomas Wallace is a programmer/analyst in the Systems Evaluation Department, with over 10 years of professional experience. Mr. Wallace has had sole responsibility for designing and implementing the interactive-flow-designer, an extension of LOGICFLOW which allows the user to design software graphically by specifying simple component structures and interconnection data. The program continually updates and displays the design in flowchart form on a CRT, providing visual feedback to the software design. He has also developed tools for, and participated in, verification and validation of numerous Minuteman programs. Mr. Wallace has a B.S. and M.S. in physics from the University of Missouri (1965, 1967).

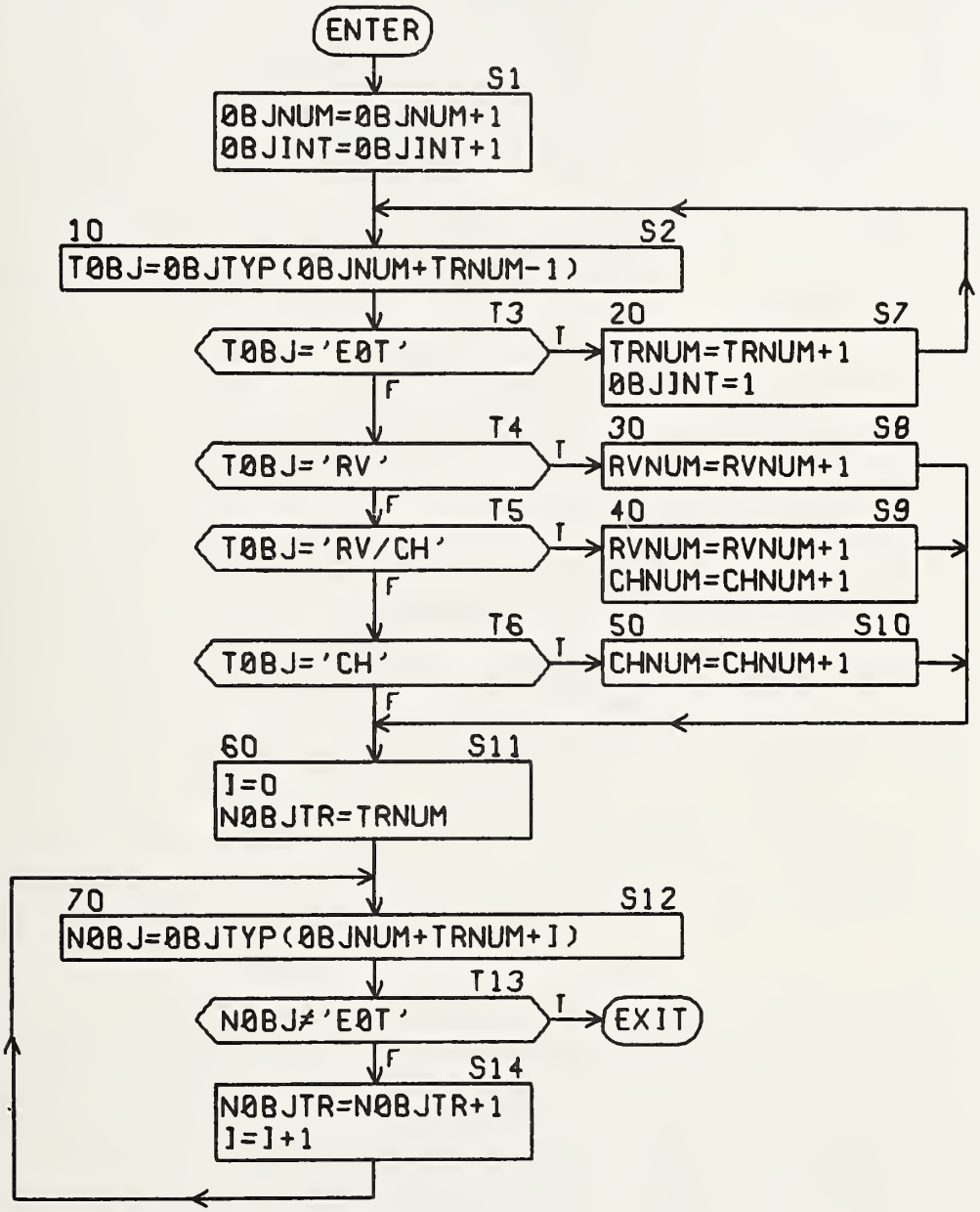
```
PROCEDURE DL EXAMPLE
    IF(CONDO)
        CASE VAR
        CASEIF(1,2)
            LOOP
                TEXT1:
            WHILE(COND1) REPEAT
            TEXT2:
        CASEIF(3,7)
            LOOP WHILE(COND2)
                TEXT3:
            REPEAT
        CASEIF()
            TEXT4:
        ENDCASE
    ELSE
        IF(COND1)
            TEXT5:
        ELSE
            LOOPFOR(I=1,7)
                TEXT6:
            REPEAT
        ENDIF
    ENDIF
    EXITPROC
ENDPROC
```

PROCEDURE DL_EXAMPLE



```
C***THIS ROUTINE UPDATES OBJECT INFORMATION AT
C****THE DEPLOYMENT OF AN OBJECT
      SUBROUTINE UPDATE
      IMPLICIT INTEGER(A-Z)
      DIMENSION OBJTYP(2)
C
      OBJNUM = OBJNUM + 1
      OBJINT = OBJINT + 1
   10 TOBJ = OBJTYP(OBJNUM + TRNUM - 1)
      IF(TOBJ.EQ.'EOT') GOTO 20
      IF(TOBJ.EQ.'RV') GOTO 30
      IF(TOBJ.EQ.'RV/CH') GOTO 40
      IF(TOBJ.EQ.'CH') GOTO 50
      GOTO 60
   20 TRNUM = TRNUM + 1
      OBJINT = 1
      GOTO 10
   30 RVNUM = RVNUM + 1
      GOTO 60
   40 RVNUM = RVNUM + 1
   50 CHNUM = CHNUM + 1
   60 I = 0
      NOBJTR = TRNUM
   70 NOBJ = OBJTYP(OBJNUM + TRNUM + I)
      IF(NOBJ.NE.'EOT') GOTO 100
      NOBJTR = NOBJTR + 1
      I = I + 1
      GOTO 70
C
  100 CONTINUE
      RETURN
      END
```

SUBROUTINE UPDATE
***THIS ROUTINE UPDATES OBJECT INFORMATION AT
****THE DEPLOYMENT OF AN OBJECT



FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . CODE INPUT
. . . FORTRAN
. . . FORTRAN IV (G1)
. . . BAL
. . . ASSEMBLY LANGUAGE
. . VHLL INPUT
. . . DESIGN LANGUAGE
. . . DL
FUNCTION
. TRANSFORMATION
. . EDITING
. . TRANSLATION
. . FORMATTING
. . RESTRUCTURING
. STATIC ANALYSIS
. . ERROR CHECKING
. . STRUCTURE CHECKING
. . COMPLEXITY MEASUREMENT
. . AUDITING
OUTPUT
. USER OUTPUT
. . GRAPHICS
. . . FLOW CHARTS
. . . DESIGN CHARTS
. . LISTINGS
. MACHINE OUTPUT
. . SOURCE CODE OUTPUT
. . . FORTRAN
. . . JOVIAL

CONTACT: DARIO DE ANGELIS, LOGICON, 255 W. FIFTH ST., PO BOX 471, SAN PEDRO, CA, 90733, USA, 213-831-0611
ROBERT J. GALVAN, LOGICON, 255 W. FIFTH ST., PO BOX 471, SAN PEDRO, CA, 90733, USA, 213-831-0611
ROGER U. FUJII, LOGICON, 255 W. FIFTH ST., PO BOX 471, SAN PEDRO, CA, 90733, USA, 213-831-0611

SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY (SREM) and the REQUIREMENTS ENGINEERING AND VALIDATION SYSTEM (REVS)

1. INTRODUCTION

Software Requirements Engineering Methodology (SREM) was developed in response to continuing and increasing difficulties in developing complex, large, real-time software for Ballistic Missile Defense (BMD) systems in the early 1970s. SREM is a formal, step-by-step process for defining data processing requirements and is not limited to BMD software applications. It provides the means to thoroughly evaluate the adequacy of system requirements towards the goal of attaining good software specifications for any system prior to design and coding. Its goal is to reduce software development cost and schedule risk.

2. SUMMARY OF SREM/REVS

SREM is a formal, integrated approach to requirements engineering activities. For software development from system requirements, it begins when the system requirements analysis has identified the system functions, the interfaces between the subsystems (at least on the functional level), the system operating rules (conditional statements impacting when and in what sequence the functions are performed), and the top level system requirements allocated to the data processor. It may also be applied at any time to verify the adequacy of existing software requirements. During the SREM activities, many problems are identified which require customer answers as to how the system is intended to function under certain circumstances. These answers are typical of processing logic not recognized as needed until final testing uncovers the fact that these circumstances have not been addressed in the software design.

In addition to the step-by-step requirements engineering techniques, SREM includes a machine-processable "English-like" Requirements Statement Language (RSL) and a Requirements Engineering and Validation System (REVS) which provides automated tools for the requirements engineer.

The SREM approach to attaining an explicit requirement specification is grounded in the use of the RSL. RSL is a formal, structured language which overcomes the shortcomings of English in stating requirements. Thus, the precise meaning of each language concept is fixed and documented to assure unambiguous interpretation of specifications using this language.

A variety of requirements analysis tools exist under REVS. Among these are an interactive graphics package, a static analyzer to assure consistency and completeness of information throughout the data base, and an automated simulation generator and execution package which aids in the study of dynamic interactions of the various requirements. Reports and analyses for engineering or management support are generated through the use of the analysis tools.

A key consideration in the SREM approach is that all the steps, including simulations, use a common requirements data base. This is necessary, since many individuals are continually adding, deleting, and changing information about requirements for the data processing system. This centralization allows both the requirements engineers and the analysis tools to work from a common, controllable baseline.

3. SCENARIO OF SREM/REVS DEMONSTRATION

The formal foundations of SREM are briefly described and the advantages of applying the methodology during the development of software are explained. The basic fundamentals of the RSL are presented and the most commonly used elements of the language are defined (e.g., subsystems, entity classes, input/output interfaces, data, and requirements networks called R_NETS).

The use of these elements in construction of a data base is described in terms of a sample problem, the College Curriculum Scheduling System (CCSS). A system level description of CCSS is presented with a brief explanation of the methods which were employed to build the CCSS data base using REVS.

The REVS software is executed on the VAX 11/780 at the BMDATC Advanced Research Center (ARC) in Huntsville, Alabama, processing the CCSS data base. An Intelligent Systems Corporation (ISC) color graphics terminal is connected via a 300 baud modem to demonstrate both the character and graphics mode operations of REVS. A TELERAY model 100 CRT terminal is also connected at 300 baud to provide system monitoring and to support the REVS demonstration. The REVS execution consists of demonstration of the three operations which are used most often--the language translator (RSL), the static analyzer (RADX), and the interactive graphics (RNETGEN). RADX demonstrates methods for listing selected portions of the data base in various formats and techniques for detecting errors in the requirements. RSL is applied to enter new information into the data base and to modify existing data to correct errors. RNETGEN is executed to present illustrations of R_NET structures, to define new structures, and to modify existing ones (An R_NET is the logic flow diagram of the processing required to respond to the stimuli of input MESSAGEs to the DP). In general, the REVS demonstration highlights the key operations of the system for an audience assumed to be familiar with the process of defining requirements, but with little or no knowledge of the SREM approach.

4. SREM/REVS LITERATURE

1. M.W. Alford, "A Requirements Engineering Methodology for Real-time Processing Requirements," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, Jan. 1977, pp. 60-69.

2. T. E. Bell, D. C. Bixler, and M. E. Dyer, "An Extendable Approach to Computer-Aided Software Requirements

Engineering," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, Jan. 1977, pp. 49-60.

3. C. G. Davis and C. R. Vick, "The Software Development System," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, Jan. 1977, pp. 69-84.

4. M. W. Alford, "Software Requirements Engineering Methodology (SREM) at the Age of Two." COMPSAC 78 Proceedings, pp. 332-339.

5. M. W. Alford, "Software Requirements Engineering Methodology (SREM) at the Age of Four," COMPSAC 80 Proceedings, pp. 866-874.

6. M. W. Alford and I. F. Burns, "R-NETS: A Graph Model for Real-Time Software Requirements," Symposium on Computer Software Engineering Polytechnic Institute of New York, April 20-22, 1976.

5. SREM/REVS DEMONSTRATORS

R.H. Hoffman

Mr. Hoffman earned his BS degree in Mathematics at Florida State University in 1968. Bob has been with TRW since graduation, where he has been involved in the development of a variety of different types of software for the Apollo Project in Houston, the Site Defense Project in Redondo Beach, and the Ballistic Missile Defense program in Huntsville. He is currently manager of the Application Software Section for TRW at the Huntsville Laboratory.

R. P. Loshbough

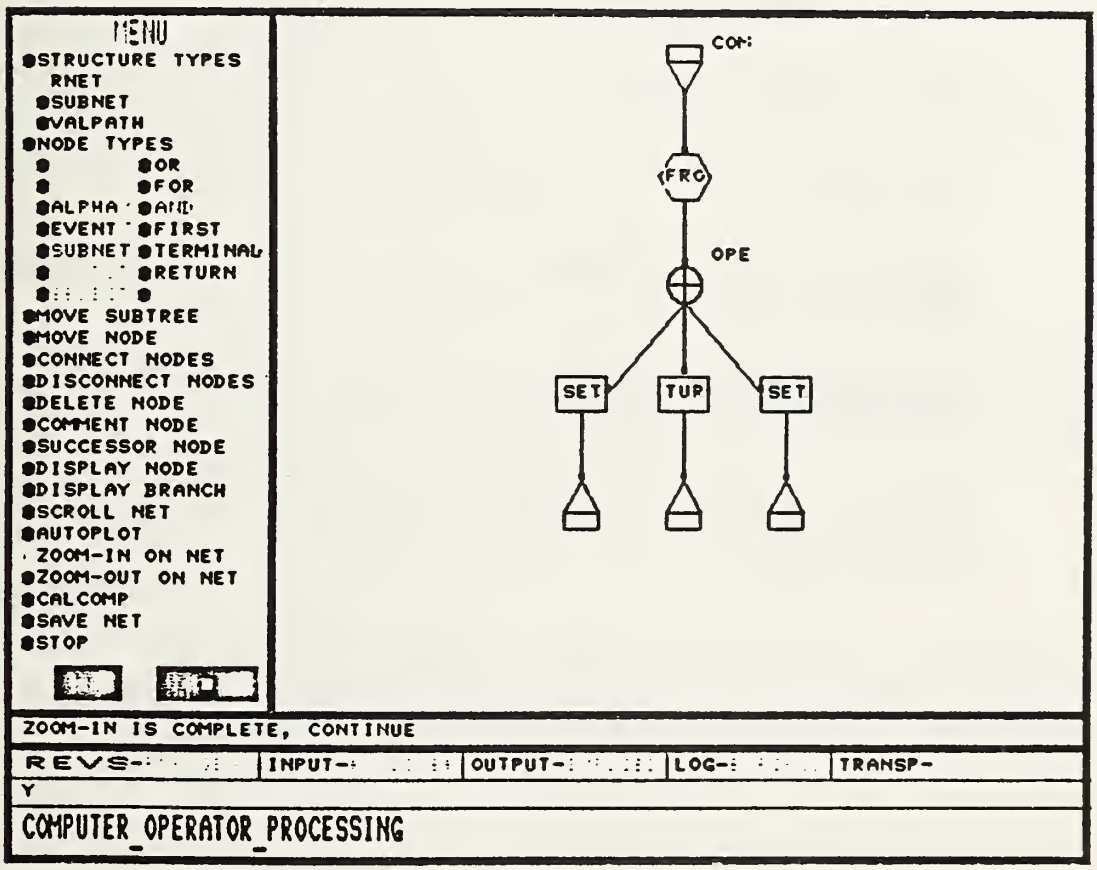
Mr. Loshbough has a BS degree in Electrical Engineering from the University of Toledo and an MBA from Babson Institute. Bob has 18 years experience in software development, and has been with TRW for 2 years. He is currently manager of the Requirements Engineering Section for TRW at the Huntsville Laboratory.

R. W. Smith

Mr. Smith holds a BS degree in Computer Science from the University of Southern Mississippi. Wayne is currently manager of the Software Development Department for TRW at the Huntsville Laboratory. He was previously involved in the development of REVS as principal investigator for the Requirements Analysis and Data Extraction (RADX) function.

```
>RADX (* PERFORM RQMTS AND DATA EXTRACTION OPERATION *).
       RADX (* PERFORM RQMTS AND DATA EXTRACTION OPERATION *).
0XX 001  FUNCTION RADX     INITIATED.  **********************************
  *-*-* ENTER RADX,  DATE = 30MAR-81,  TIME = 07:42:56 *-*-*
 [RADX COMMAND=
>LIST SUBSYSTEM (* SUBSYSTEMS EXTERNAL TO CCSS DP SYSTEM *).
 LIST SUBSYSTEM (* SUBSYSTEMS EXTERNAL TO CCSS DP SYSTEM *).
 ------------------------------------------------------------

    SUBSYSTEM:  ADMIN_OFC.
         CONNECTED TO:
              INPUT_INTERFACE:  FROM_ADMIN_OFC
              OUTPUT_INTERFACE:  TO_ADMIN_OFC.

    SUBSYSTEM:  MULTIPLEXER.
         CONNECTED TO:
              INPUT_INTERFACE:  FROM_MULTIPLEXER
              OUTPUT_INTERFACE:  TO_MULTIPLEXER.

    SUBSYSTEM:  OPERATOR.

    SUBSYSTEM:  REGISTRAR.

 [RADX COMMAND=
>APPEND ALL NONE (* ABBREVIATE LISTS *).
 APPEND ALL NONE (* ABBREVIATE LISTS *).
 ---------------------------------------

 [RADX COMMAND=
>LIST INPUT_INTERFACE (* LIST INPUT INTERFACES TO CCSS *).
 LIST INPUT_INTERFACE (* LIST INPUT INTERFACES TO CCSS *).
 ---------------------------------------------------------

    INPUT_INTERFACE:  FROM_ADMIN_OFC.

    INPUT_INTERFACE:  FROM_MULTIPLEXER.

    INPUT_INTERFACE:  FROM_OPERATOR.

    INPUT_INTERFACE:  FROM_REGISTRARS_OFC.

 [RADX COMMAND=
>APPEND ALL STRUCTURE (* INCLUDE STRUCTURE IN LISTS *).
 APPEND ALL STRUCTURE (* INCLUDE STRUCTURE IN LISTS *).
 -----------------------------------------------------

 [RADX COMMAND=
>LIST COMPUTER_OPERATOR_PROCESSING (* LIST AN R_NET *).
 LIST COMPUTER_OPERATOR_PROCESSING (* LIST AN R_NET *).
 -----------------------------------------------------

    R_NET:  COMPUTER_OPERATOR_PROCESSING
    (*400*).
         STRUCTURE:
            INPUT_INTERFACE:  FROM_OPERATOR
            CONSIDER  DATA:  OPERATOR_MSG_TYPE
            IF (REESTABLISH_REGISTRAR_TERMINAL)
               ALPHA:  TURN_OFF_INPUT_PROHIBITED_FLAG
               TERMINATE
            OR (SET_SEMESTER)
               ALPHA:  SET_NEW_SEMESTER
               TERMINATE
            OR (SET_YEAR)
               ALPHA:  SET_NEW_YEAR
               TERMINATE
            END
         END.
```

```
 [RADX COMMAND=
>ANALYZE DATA_FLOW COMPUTER_OPERATOR_PROCESSING.
 ANALYZE DATA_FLOW COMPUTER_OPERATOR_PROCESSING.

*-*-*-* ANALYZE DATA FLOW FOR R_NET:  COMPUTER_OPERATOR_PROCESSING
*ERROR 2641 ELEMENT ON CONSIDER OR DOES NOT HAVE TYPE ATTRIBUTE
      DATA:  OPERATOR_MSG_TYPE
   * ERROR DETECTED AT OR-NODE   DATA:  OPERATOR_MSG_TYPE
   * PRECEDED BY INPUT_INTERFACE:  FROM_OPERATOR
   * PRECEDED BY R_NET:  COMPUTER_OPERATOR_PROCESSING


*ERROR 2643 ELEMENT ON CONSIDER OR DOES NOT HAVE RANGE ATTRIBUTE
      DATA:  OPERATOR_MSG_TYPE
   * ERROR DETECTED AT OR-NODE   DATA:  OPERATOR_MSG_TYPE
   * PRECEDED BY INPUT_INTERFACE:  FROM_OPERATOR
   * PRECEDED BY R_NET:  COMPUTER_OPERATOR_PROCESSING


*ERROR 2820 INFORMATION PASSING INPUT_INTERFACE NOT USED.
      DATA:  NEW_TEAR_IN.
```



FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . VHLL INPUT
. . . REQUIREMENTS STATEMENT LANGUAGE
. . . RSL
FUNCTION
. STATIC ANALYSIS
. . COMPLETENESS CHECKING
. . CONSISTENCY CHECKING
. . MANAGEMENT
. . . DATA BASE MANAGEMENT
. DYNAMIC ANALYSIS
. . SIMULATION
OUTPUT
. USER OUTPUT
. . DIAGNOSTICS
. . USER-ORIENTED TEXT
. . . REPORTS
. . GRAPHICS
. . LISTINGS

IMPLEMENTATION LANGUAGE: FORTRAN, PASCAL

TOOL PORTABLE: YES

TOOL AVAILABLE: YES, PUBLIC DOMAIN: YES

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): NO RESTRICTIONS WITHIN THE USA

TOOL SUPPORTED: YES, TOOL SUPPORT: TRW DEFENSE AND SPACE SYSTEMS GROUP, HUNTSVILLE, AL

CONTACT: ROBERT H. HOFFMAN, TRW INC., HUNTSVILLE FACILITY, 7702 GOVERNORS DRIVE WEST, HUNTSVILLE, ALABAMA, 35805, USA, 205-837-3950

SDP: A COMPUTERIZED TOOL FOR SYSTEM DESIGN AND MAINTENANCE

Proposal for Demonstrating SDP at the 5th ICSE

Nancy Linden and Moshe Yavne

1. DESCRIPTION OF SDP

SDP is a very high level language (VHLL) that applies Software Engineering methodologies such as Top-down design, structured design, and data abstraction to the design of systems. The designer expresses his ideas in a structured English-like language. SDP produces documents which display the design in a clear and readable manner so that all those involved can easily understand the solution. SDP formats the design modules, providing full cross-referencing of the modules and data items, and a tree representing the hierarchical sequence of referencing (calling tree). In addition, SDP provides a mechanism for designing data structures using abstract data typing, and for designing the control of synchronization between processes. Additional features include text modules, inclusion of external modules, interface definition, user defined keywords, and parameterized module names. SDP performs several consistency checks on the design such as proper use of control primitives, and interface between modules.

The functional specification for SDP was defined under research at the University of California at Los Angeles, and the processor was developed by MAYDA, Software Engineering. SDP is currently used throughout Israel, including the Ministry of Defense, and Israel Aircraft Industries. It has been used both for the development of new systems, and for documenting old systems for maintenance.

2. SCENARIO OF THE SDP DEMONSTRATION

There will be two types of demonstrations of SDP. One will be a prepared demonstration where the demonstrator will choose an appropriate example and will invoke SDP to produce the design document. The second will be where the user would define a sample design in pseudo-code, and the demonstrator would activate SDP using the user's input.

3. LITERATURE

A. SDP REFERENCE MANUAL
A detailed description of all the features provided by SDP, including a sample design. Appendices provide the error messages, and a detailed explanation of abstract data types.

B. PROGRAMMER'S GUIDE
Describes the motivation behind the development of SDP, the methodologies used by SDP and a description of how to design with SDP.

C. PAPERS

"Software Development Processor: A Tool for Software Design," Nancy May Linden, Masters Thesis, University of California, 1976, Los Angeles.

"SDP, A Tool for Software Design and Maintenance," Nancy Linden, Mayda Software Engineering, Rehovot, Israel, 1981.

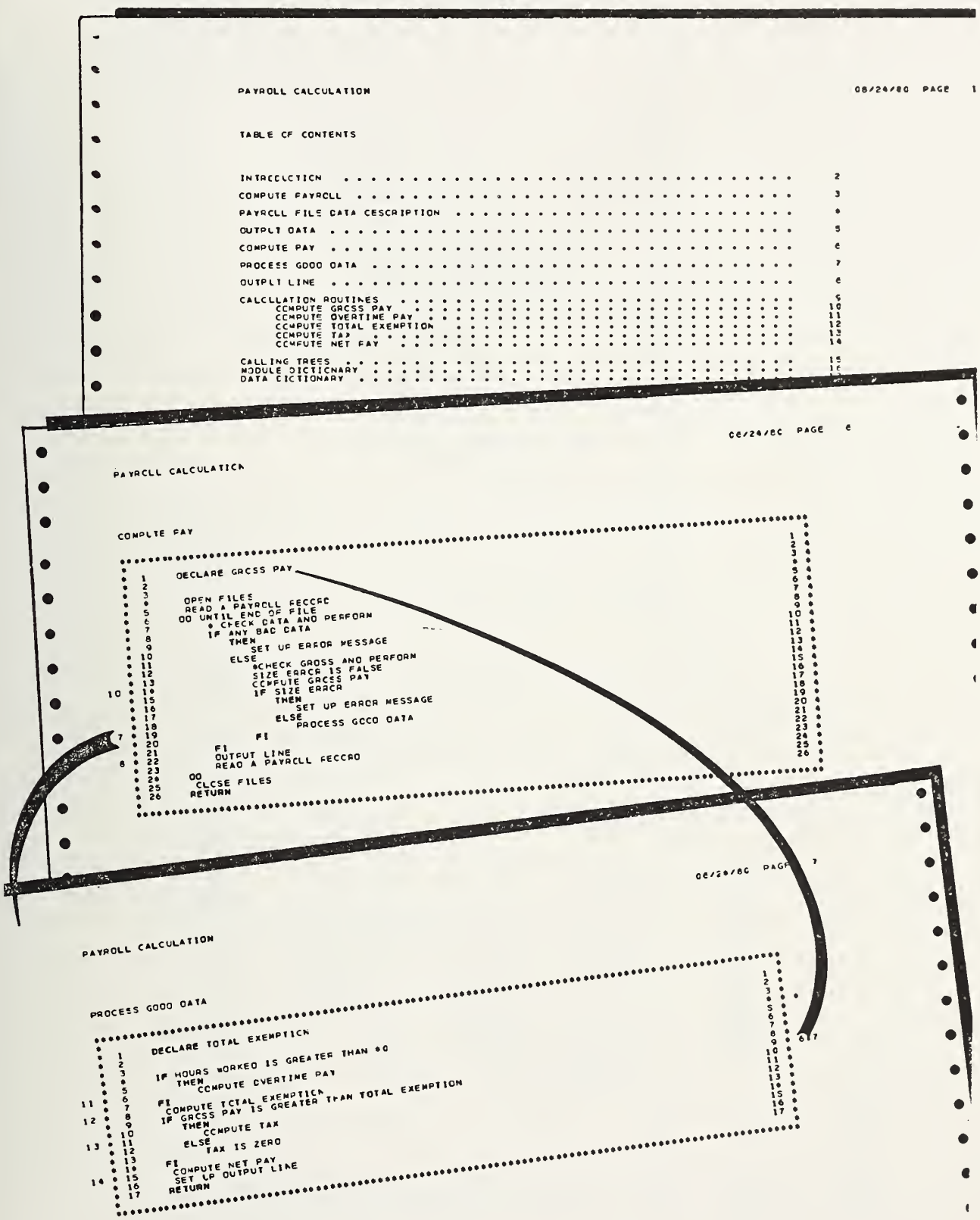
4. STATION, DAY AND TIME

Station 7, Wednesday March 11, from 9-12 and 2-5.

5. THE DEMONSTRATORS

Nancy Linden is a member of the managing staff of Mayda Software Engineering, Israel, where she is responsible for the development of Software Engineering Tools. She received her Master's in Computer Science at the University of California at Los Angeles, where she originally defined the functional specification of SDP. Prior to joining Mayda, she was involved in the areas of compiler construction, in the U.S., and data networks in Holland and Israel.

Moshe Yavne is a member of the managing staff of Mayda Software Engineering, Israel. He is responsible for research and development. He received his Master's in Computer Science at the University of California at Los Angeles, where he performed research on the Graph Model of Computation for the Atomic Energy Commission. Prior to joining Mayda, he was involved in the Voyageur software effort at Jet Propulsion Laboratory, and in data communication networks in Holland.



PAYROLL CALCULATION 08/24/80 PAGE 1

TABLE OF CONTENTS

08/24/80 PAGE 6

PAYROLL CALCULATION

COMPUTE PAY

```
1    DECLARE GROSS PAY
2
3     OPEN FILES
4     READ A PAYROLL RECORD
5     DO UNTIL END OF FILE
6        * CHECK DATA AND PERFORM
7        IF ANY BAD DATA
8           THEN
9              SET UP ERROR MESSAGE
10          ELSE
11             *CHECK GROSS AND PERFORM
12             SIZE ERROR IS FALSE
13             COMPUTE GROSS PAY
10 14          IF SIZE ERROR
15                THEN
16                   SET UP ERROR MESSAGE
17                ELSE
7 18                PROCESS GOOD DATA
19             FI
20       FI
21       OUTPUT LINE
8 22     READ A PAYROLL RECORD
23    OD
24    CLOSE FILES
25    RETURN
26
```

08/24/80 PAGE 7

PAYROLL CALCULATION

PROCESS GOOD DATA

```
1    DECLARE TOTAL EXEMPTION
2
3    IF HOURS WORKED IS GREATER THAN 40
4       THEN
5          COMPUTE OVERTIME PAY
6
11 7    FI
12 8    COMPUTE TOTAL EXEMPTION
9       IF GROSS PAY IS GREATER THAN TOTAL EXEMPTION
10         THEN
13 11         COMPUTE TAX
12         ELSE
13            TAX IS ZERO
14 15   FI
16      COMPUTE NET PAY
17      SET UP OUTPUT LINE
        RETURN
```

```
LINE DRIVER


QUEUE  IS:     QUEUE NOT EMPTY
               GET NEXT BUFFER_CONTROL_BLOCK
               APPEND BUFFER_CONTROL_BLOCK TO QUEUE


        DECLARE QUEUE_LENGTH AS COUNTER
        DECLARE BUFFER_CONTROL_BLOCK
        DECLARE HEAD AS A POINTER
        DECLARE TAIL AS A POINTER


     OP QUEUE NOT EMPTY
         RETURNS TRUE OR FALSE
        IF QUEUE_LENGTH = 0
           THEN
              RETURN FALSE
           ELSE
              RETURN TRUE
        FI

     OP GET NEXT BUFFER_CONTROL_BLOCK
        IF QUEUE_LENGTH NOT 0
           THEN
              REMOVE HEAD BUFFER_CONTROL_BLOCK
              DECREMENT QUEUE_LENGTH
              RETURN BUFFER_CONTROL_BLOCK
           ELSE
              ERROR
        FI

     OP APPEND BUFFER_CONTROL_BLOCK TO QUEUE
         INSERT BUFFER_CONTROL_BLOCK AT TAIL OF QUEUE
         INCREMENT QUEUE_LENGTH
```

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . VHLL INPUT
FUNCTION
. TRANSFORMATION
. . FORMATTING
. STATIC ANALYSIS
. . STRUCTURE CHECKING
. . CROSS REFERENCE
. . SCANNING
OUTPUT
. USER OUTPUT
. . GRAPHICS
. . . HIERARCHICAL TREE
. . LISTINGS
. . USER-ORIENTED TEXT
. . . DOCUMENTATION

IMPLEMENTATION LANGUAGE: FORTRAN 66

TOOL PORTABLE: YES, TOOL SIZE: 64 KB

TOOL AVAILABLE: YES, PUBLIC DOMAIN: NO

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): LICENSE

TOOL SUPPORTED: YES, TOOL SUPPORT: MAYDA SOFTWARE ENGINEERING

CONTACT: MAYDA SOFTWARE ENGINEERING, PO BOX 1389, REHOVOT, 76113, ISRAEL, Ø54-58534

SOFTOOL 80™

A METHODOLOGY AND A COMPREHENSIVE SET OF TOOLS FOR SOFTWARE MANAGEMENT, DEVELOPMENT AND MAINTENANCE.

1. INTRODUCTION

SOFTOOL 80 is an integrated methodology and supporting tools for software management, development and maintenance. It is a portable system currently supported on SEL, DEC, DG and IBM computers.

SOFTOOL 80 consists of several major releases addressing: 1) Programming Environment, 2) Change and Configuration Control, 3) Design, and 4) Requirements. At the 5th International Conference on Software Engineering in San Diego, Calif., March 1981, we will be demonstrating Release 1: Programming Environment.

SOFTOOL 80 is a commercially available product fully supported with manuals, interactive tutorials, courses and maintenance. It is a proprietary product of Softool Corporation, Goleta, California. It is currently in use in over 35 different installations.

2. SUMMARY

Conceptually, SOFTOOL 80, Release 1, allows a user to interactively create in an 'application generator mode' a substantial portion of a new application, typically, over 50% of the required code. The user then, with the aid of an elaborate collection of software tools, completes the program and generates a deliverable product. The areas supported with existent tools include: structured programming at a level that matches design documents, extensive diagnostics, code auditing, portability, documentation, tracing, testing, time and space optimization. Management visibility, standards and quality control are given explicit support.

Experience in the use of SOFTOOL 80, Release 1, indicates that fivefold improvements, over conventional approaches, in the amount and quality of the software created are readily attained.

3. A SCENARIO FOR THE DEMONSTRATION

The demonstration will consist of two parts.

The first part is a summary slide presentation outlining the methodology and tools of SOFTOOL 80. The presentation will run for about 30 minutes. This presentation will be given several times according to a preposted schedule.

The second part is a hands on demonstration at the terminals to take place between slide presentations. Participants will be encouraged to sit at the terminals and exercise various components of SOFTOOL 80.

A color outline of a SOFTOOL 80 scenario will be passed out to those attending a demonstration.

4. SOFTOOL 80 LITERATURE

Reference Manuals

There is a comprehensive reference manual for each SOFTOOL 80 component.

Interactive Tutorials

There is a growing number of interactive tutorials to assist in the effective training of new users.

Technical Notes

There is a growing library of short notes discussing specific issues in the SOFTOOL 80 context. Areas addressed by existing notes include: structured programming, error resistant programs, instrumentation, flow analysis and portability.

Technical Reports

There is a growing library of technical reports providing thorough discussion of a number of specific topics in the SOFTOOL 80 context. Areas addressed by existing reports include: testing, optimization, portability and interactives. (One of these reports entitled 'A Pragmatic Approach to Portable Software' appeared in Computerworld in-Depth report, April 14,80 and in Mini Micro Systems, May,80)

Product Brochures

Marketing brochures describing the various SOFTOOL 80 components

5. STATION, DAY AND TIME

STATION: 8,10

DAY: WEDNESDAY, MARCH 11, 1981

TIME: 9 a.m. To: 5:30 p.m.

6. DEMONSTRATORS

RICHARD HUG

Mr. Hug received his B.S. degree in Computer Science in 1975 from the University of California, Santa Barbara. His current interests are in software methodology and tools. He is a senior project manager at Softool Corporation.

THOMAS STRELICH

Mr. Strelich received his B.S. degree in Biology in 1977 from Calif. State College, Bakersfield, and his M.S. degree in Computer Science in 1980 from California Polytechnic State University, San Luis Obispo. His current interests are in programming languages and systems. Mr. Strelich is a member of the technical staff of Softool Corporation.

MICHAEL RESNICOW

Mr. Resnicow received his B.S. degree in Electrical Engineering in 1975 from Tufts University, Boston Massachusetts, and his M.B.A. at Monmouth College, New Jersey. His interests include software methodology and tools and optimization technology. He is currently Software Sales Manager at Systems Engineering Laboratories, Incorporated in Fort Lauderdale, Florida.

ROBERT AHOLA

Mr. Ahola received his A.A.S. degree in chenistry in 1963 from University of Minnesota and further studies at Columbia and State University of New York at Buffalo in Engineering and Computer Science. His interests include artificial language and development methodologies. His current position is Technical Software Support Manager at Systems Engineering Laboratories, Incorporated in Fort Lauderdale, Florida.

SOFTOOL 80™, Release 1, is an integrated collection of over 20 major tools. Here we present some examples.

Sample Explosion of Interfaces

```
                ***  EXPLOSION FOR MAIN    ***

MAIN        M
     E.XI         U
     ENTER        E
     SUB1         M
          COM2          C
          COM1          C
          INSERT        U
          SUB3          U
          SUB4          M
               SUB2          M
                    SUB3          U
                    SUB4          M
                         **RECURSION**
               SUB5          M
                    SUB3          U
                    SUB6          M
          SEARCH        M
               ENTER         E
               COM2          C
               INSERT        U
               B3            M
     SUB2         M
          **REPEATED NODE**
```

Sample Test Coverage Report - Routine level

```
**********************************************************************
*                                                                    *
*       T E S T   C O V E R A G E   D O C U M E N T A T I O N        *
*                                                                    *
**********************************************************************
```

TEST COVERAGE PROFILE

TOTAL ROUTINES IN SYSTEM: 14

	NUMBER OF ROUTINES EXERCISED	% OF TOTAL ROUTINES	NUMBER OF ROUTINE EXECUTIONS	TEST EFFECTIVENESS (NUMBER EXECUTIONS/ % TOTAL ROUTINES)
THIS RUN:	4	28.57	46	.02
ALL RUNS:	5	35.71	79	.02
DIFFERENT ROUTINES IN THIS RUN:	1	7.14	46	.06

Test coverage accomplished

This is an indication of the cost effectiveness of the testing activity; the lower the better

Sample Optimization Report - Statement level

```
**********************************************************************
*                                                                    *
*        O P T I M I Z A T I O N   D O C U M E N T A T I O N         *
*                                                                    *
**********************************************************************

                              ROUTINE: SEARCH

                          TOTAL TIME PROFILE
                            SINGLE RUN

STATEMENT NUMBER      TOTAL TIME (MSEC)      % OF TOTAL TIME

             26              56.320                35.53
             11              38.408                24.23
              2              28.931                18.25
              9              13.045                 8.23
              6              10.595                 6.68
             41               5.028                 3.17
              4               1.550                  .98
             27                .896                  .57
             24                .576                  .36
             33                .504                  .32
             12                .496                  .31
              7                .421                  .27
             22                .288                  .18
```

Note that the first 3 statements account for 78% of the total time. That is, 7% of the statements in the routine account for 78% of the time. This is typical

The output is ordered according to time consumption

Sample Error Detection Report through Dynamic Flow Analysis:

```
E301    FLOW ERROR
        ROUTINE:        STRD
        STRUCTURE:      STRUC
        TRANSITION:     UNDEF/READ
        RECORD:         6
        FIELD:          2
        CURRENT # OF RECORDS:   5
```

The above information indicates that routine STRD, which reads fields of records, attempted to read an undefined field. The attempt was to read field 2 of record 6 of the structure STRUC. However, note that the current number of records in STRUC is only 5, thus record 6 is undefined.

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . CODE INPUT
. . VHLL INPUT
FUNCTION
. TRANSFORMATION
. . INSTRUMENTATION
. . TRANSLATION
. . FORMATTING
. STATIC ANALYSIS
. . MANAGEMENT
. . . CONFIGURATION MANAGEMENT
. . DATA FLOW ANALYSIS
. . STRUCTURE CHECKING
. . COMPLEXITY MEASUREMENT
. . AUDITING
. . COMPARISON
. . COMPLETENESS CHECKING
. . SCANNING
. . INTERFACE ANALYSIS
. DYNAMIC ANALYSIS
. . COVERAGE ANALYSIS
. . TRACING
. . TUNING
. . TIMING
OUTPUT
. USER OUTPUT
. . TABLES
. . LISTINGS
. . DIAGNOSTICS
. . USER-ORIENTED TEXT
. . . DOCUMENTATION
. MACHINE OUTPUT
. . SOURCE CODE OUTPUT

IMPLEMENTATION LANGUAGE: FORTRAN, TOOL PORTABLE: YES, TOOL SIZE: 2K - 27ØK BYTES, TOOL AVAILABLE: YES, PUBLIC DOMAIN: NO, RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): PROPRIETARY PRODUCT, TOOL SUPPORTED: YES, TOOL SUPPORT: SOFTOOL CORPORATION

CONTACT: SOFTOOL CORPORATION, 34Ø S. KELLOGG, GOLETA, CA, 93117, USA, 8Ø5-964-Ø56Ø

ITB:
Interactive Test Bed

A Demonstration for the Software Tool Fair at the
Fifth International Conference on Software Engineering,
March 9-12, 1981

James B. Henderson, Edward F. Miller, Jr.

1. Introduction

This is a description of demonstrations of the **Interactive Test Bed System** (ITB) at the Software Tool Fair of the Fifth International Software Engineering Conference in San Diego, March 9-12, 1981. The ITB system has been developed by Software Research Associates.

2. Summary of ITB

The Interactive Test Bed (ITB) system is a system for performing quality assurance analysis of modular software systems.

Invocation of two macros is sufficient to create a test bed for testing a given routine. Once the test bed is created, ITB allows the user interactive access to any of the variables in the commons and parameter list in the test source which was analyzed by the ITB set up program. The user may alter or examine any of these variables. The user is empowered to execute a series of tests of the program, to manipulate and examine the values of variables as well as consult the (C1) coverage report between executions. The NOT HIT command pinpoints the segments which have not yet been executed. By saving copies of the environment of variables, ITB can help process complex tree-structured sets of test cases. Additionally, ITB is useful in the software testing documentation process.

The ITB system to be demonstrated processes pure-FORTRAN or SRTRAN (a structured extension of FORTRAN) programs. However, the concepts of the ITB extend beyond FORTRAN to any language or environment.

3. Demonstration Scenario

As indicated in the summary information, the Interactive Test Bed (ITB) is a system for the on-line testing of FORTRAN programs. The general demonstration will consist of one of our staff using the test bed to exercise some sample programs. The test beds for each example will have been prepared in advance. What the Tool Fair attendees will be able to watch is the process of setting up input values using the test bed, running of tests with ITB, examining coverage reports generated by ITB, examining output results, and repeating the process of altering data, re-running tests, checking coverage and output, etc.

Scripts of test sessions can also be run on-line in a batch mode to speed data entry, yet still produce a trace of the whole ITB session.

If time permits Fair attendees may be able to try running the test bed examples themselves. The difficulty of entering into the system an arbitrary program supplied by the user precludes the possibility of demonstrating ITB on such an arbitrary program at the Tool Fair.

4. ITB References

The following documents make up the reference library for ITB. Documents marked with an asterisk will be available at the Tool Fair.

Tom Mapp and Robert Schulman, "ITB Command Reference Manual -- Data General SRTRAN Version", TN-690/4, Software Research Associates, August 1980.

* Tom Mapp, "ITB Sample Outputs," TN-801, Software Research Associates, February 1981.

* "ITB Fact Sheet," Software Research Associates, September, 1980.

* "ITB Command Summary," Software Research Associates, February 1981.

* "SofTest Methodology Description / Fact Sheet," Software Research Associates, February 1981.

5. Demonstration Location and Time

Station 9, Wednesday, March 11, from 9:00 am until 4:00 pm, at specific times to be posted at the station (4 - 6 demonstrations throughout the day).

6. The Demonstrators

Dr. EDWARD F. MILLER, JR., is Technical Director of Software Research Associates, San Francisco, California, a firm devoted to advanced computer technology and software applications. His interests include software engineering management, software testing technology, software maintenance technology, automated tool design and computer architecture.

Dr. Miller was previously Director of the Software Technology Center, Science Applications, Inc., San Francisco, and Director of the Program Validation Project at General Research Corporation, Santa Barbara, California. He received a BSEE at Iowa State University in 1962, an M. S. in Applied Mathematics at the University of Colorado in 1964, and the Ph. D. at the University of Maryland in 1968 where he was an Instructor from 1964 to 1968.

Dr. Miller is a member of the IEEE Computer Society, the ACM, SIAM and several honorary societies. He currently serves on several technical committees and is an Associate Technical Editor of COMPUTER Magazine.

Mr. JAMES B. HENDERSON is a programmer who specializes in the development of advanced concept tools for modern Software Engineering. His

current research interest center on the development of techniques for hierarchical decomposition of programs based on their directed graph structure.

Since joining SRA in June 1979 he has been involved in a number of projects that support various Company projects. He has been responsible for production of the baseline version of SRTRAN, the Company's structured preprocessor system for Structured FORTRAN programming.

Mr. Henderson has supported the Company quality assurance program by participating in the development of a COBOL testbed system for the UNIVAC 1100 system, and through development of special versions of the COBOL Instrumentation Subsystem also for the UNIVAC 1100 environment. He is currently responsible for quality control on the entire COBOL Instrumentation Subsystem (all machine environments).

He has also participated in other Company efforts, including the development of the Semantic Update System (ISUS) series of automated software maintenance tools, and various in-house administrative processing facilities.

7. ITB Contact

For further information please contact:

Dr. Edward Miller
Software Research Associates
P. O. Box 2432
San Francisco, CA 94126
(415) 957-1441

8. Proprietary Statement

This system is proprietary to Software Research Associates. The demonstrations of the system will be made for the purpose of describing the technical capabilities of the system only. Extensive documentation of a proprietary nature is available but will not be distributed to attendees without appropriate levels of protection of proprietary rights. Information and materials that are distributed freely at the demonstrations carry no such restriction, however.

SAMPLE COMMAND SEQUENCES

```
    1                OPTIONS(INST=Y,LINWDT=71)
    2                SUBROUTINE ADD
    3                COMMON /ADDCOM/ N, A, SUM
    4                INTEGER N, I
    5                INTEGER A(10), SUM
             C
             C       THE PURPOSE OF THIS PROCEDURE IS TO ADD UP THE
             C       CONTENTS OF A(1)...A(N) AND PLACE
             C       THE TOTAL IN THE OUTPUT VARIABLE SUM
             C
             C       INITIALIZATION...
             C
    6                ISUM = 0
             C       INITIAL PROTECTION...
             C
    7                IF (N .GT. 1)                                    (2)
      ( 1)   C       THE ITERATION...
      ( 1)   C
    8 ( 1)           .  I = 1
    9 ( 1)           .  WHILE (I .LE. N)                              (3)
   10 ( 2)           .  .  ISUM = ISUM + A(I)
   11 ( 2)           .  .  I = I + 1
   12 ( 1)           .  END WHILE                                     (4)
      ( 1)   C
      ( 1)   C       ALTERNATIVE...
      ( 1)   C
   13 ( 1)           ELSE                                             (5)
   14 ( 1)           .  WRITE (10, 10) N
   15 ( 1)  10       .  FORMAT ( 22H ERROR...N HAS VALUE =    , I10)
   16                END IF                                           (6)
             C
             C       CAPTURE THE RESULT...
             C
   17                SUM = ISUM
   18                RETURN
   19                END
```

```
NUMBER OF SRTRAN STATEMENTS IN THIS MODULE IS     8
NUMBER OF FORTRAN STATEMENTS IN THIS MODULE IS   11
NUMBER OF COMMENT STATEMENTS IN THIS MODULE IS   17
TOTAL NUMBER OF STATEMENTS IN THIS MODULE IS     36
```

Figure A — Example Routine ADD

SAMPLE COMMAND SEQUENCES

```
. . * Set ECHO so that commands read from ghost file will be
. * echoed on the output file
. ECHO
. * Make PROMPT character into a question mark "?"
. PROMPT "?"
?NEW N IS 4
?SEE SUM
common ADDCOM, offset   12:      0 000000
?* SUM is set to 0
?SEE N
common ADDCOM, offset    1:      4 000004
?* N has been set to 4
?NEW A(1) IS -5
?NEW A(2) IS 305
?NEW A(4) IS 46
?* Now we can look at the values put into array A
?SEE A(1..4)
common ADDCOM, offset    2:     -5 177773
common ADDCOM, offset    3:    305 000461 "1'
common ADDCOM, offset    4:      0 000000
common ADDCOM, offset    5:     46 000056 ".'
?* Note how far right also provides us with the
?* alphanumeric interpretation of the value.
?EXEC
?SEE SUM
common ADDCOM, offset   12:    346 000532 "Z'
?STOP
```

Figure B — Setting and Viewing Input Variables

SAMPLE COMMAND SEQUENCES

```
?NEW SUM IS 35
?NEW N IS 5
?NEW A(1) IS 2
?NEW A(2) IS 4
?NEW A(3) IS 5
?NEW A(4) IS -1
?NEW A(5) IS -3
?SEE A(1..5)
common ADDCOM, offset    2:       2 000002
common ADDCOM, offset    3:       4 000004
common ADDCOM, offset    4:       5 000005
common ADDCOM, offset    5:      -1 177777
common ADDCOM, offset    6:      -3 177775
?EXEC
?SEE SUM
common ADDCOM, offset   12:       7 000007
?* Now this environment will be saved
?PUSH SAVE1
?NEW N IS -3
?NEW SUM IS 0
?* Put invalid values in array A
?NEW A(1..5) IS 0
?SEE A(1..5)
common ADDCOM, offset    2:    8224 020040 '  '
common ADDCOM, offset    3:    8224 020040 '  '
common ADDCOM, offset    4:    8224 020040 '  '
common ADDCOM, offset    5:    8224 020040 '  '
common ADDCOM, offset    6:    8224 020040 '  '
?EXEC
 ERROR...N HAS VALUE =         -3
?* Recall old environment
?FROM STACK SAVE1
?SEE SUM
common ADDCOM, offset   12:       7 000007
?SEE A(1..5)
common ADDCOM, offset    2:       2 000002
common ADDCOM, offset    3:       4 000004
common ADDCOM, offset    4:       5 000005
common ADDCOM, offset    5:      -1 177777
common ADDCOM, offset    6:      -3 177775
?SEE N
common ADDCOM, offset    1:       5 000005
?STOP
```

Figure C — Saving Environments

```
?TEST ADD ONE
?NEW N IS 3
?NEW A(1) IS -271
?NEW A(2) IS 84
?NEW A(3) IS 4
?SEE A(1..3)
common ADDCOM, offset    2:   -271 177361
common ADDCOM, offset    3:     84 000124 "T'
common ADDCOM, offset    4:      4 000004
?EXEC
?* How many segments did this test exercise?
?C1
Coverage for module ADD     :
   (Most recent test)    5 segments hit out of   6. C1:  83.33%
   (Previous tests)      0 segments hit out of   6. C1:    .00%
?* Which segments were not exercised?
?NOT HIT
Not hit report for ADD
   Segments not hit in most recent test:
      Segment   5 not hit
   Segments not hit in previous tests:
      Segment   1 not hit
      Segment   2 not hit
      Segment   3 not hit
      Segment   4 not hit
      Segment   5 not hit
      Segment   6 not hit
?END TEST
?TEST ADD TWO
?NEW N IS -110
?EXEC
 ERROR...N HAS VALUE =      -110
?C1
Coverage for module ADD     :
   (Most recent test)    3 segments hit out of   6. C1:  50.00%
   (Previous tests)      5 segments hit out of   6. C1:  83.33%
?END TEST
?* Merging this last test with previous tests, gives us this coverage
?C1
Coverage for module ADD     :
   (Most recent test)    3 segments hit out of   6. C1:  50.00%
   (Previous tests)      6 segments hit out of   6. C1: 100.00%
?NOT HIT
Not hit report for ADD
   Segments not hit in most recent test:
      Segment   2 not hit
      Segment   3 not hit
      Segment   4 not hit
   Segments not hit in previous tests:
     All segments hit.
?STOP
```

Figure D — Reporting C1 Test Effectiveness

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . CODE INPUT
. . . SRTRAN
. . . FORTRAN
FUNCTION
. TRANSFORMATION
. . INSTRUMENTATION
. DYNAMIC ANALYSIS
. . COVERAGE ANALYSIS
. . TRACING
OUTPUT
. USER OUTPUT
. . TABLES
. . LISTINGS
. MACHINE OUTPUT
. . SOURCE CODE OUTPUT
. . . SRTRAN
. . . FORTRAN

IMPLEMENTATION LANGUAGE: SRTRAN

TOOL PORTABLE: YES

TOOL AVAILABLE: YES, PUBLIC DOMAIN: NO

TOOL SUPPORTED: YES, TOOL SUPPORT: SOFTWARE RESEARCH ASSOCIATES

CONTACT: EDWARD F. MILLER, SOFTWARE RESEARCH ASSOCIATES, P.O. BOX 2432, SAN FRANCISCO, CA, 94126, USA, 415-957-1441

ISUS:
Interactive Semantic Update System

A Demonstration for the Software Tool Fair at the
Fifth International Conference on Software Engineering,
March 9-12, 1981

James B. Henderson, Edward F. Miller, Jr., Morton Hirschberg

1. Introduction

This is a description of demonstrations of the **Interactive Semantic Update System** (ISUS) at the Software Tool Fair of the Fifth International Software Engineering Conference in San Diego, March 9-12, 1981. The ISUS system has been developed by Software Research Associates in cooperation with the U. S. Army Ballistics Research Laboratory.

2. Summary of ISUS

ISUS is a system that combines features of a text editor, a source code control system, a configuration management system, and a static analyzer to provide an integrated facility for maintenance of complex software systems. The most important feature of ISUS is that it has the capability to perform single- and multiple-module consequence analysis, either interactively or in batch mode.

The ISUS system is designed as an interactive tool, responding to user commands, but it can also be run in a batch mode, reading from a prepared command file, for "background" runs performing the day to day chores associated with system maintenance.

The current version of ISUS processes FORTRAN programs which have been organized into a special format, called a Program Master (PM). The PM contains both the system's modules (main programs, subroutines, functions, block datas) and the system's globals (COMMON declarations of data areas to be used by many modules). ISUS contains an INCLUDE facility whereby globals can be referenced by modules.

ISUS reads in a PM at the start of a session, modifies it in response to user commands, and produces a new PM at the end of the session. ISUS also produces a trace file containing all commands issued by the user during a session. This trace file in combination with the old PM and ISUS itself typifies the changes made to the system in a readily understandable form. By saving the original PM of a system and the trace files from a series of changes made any of the intermediate versions of the system can be reproduced by running ISUS in the batch mode with the trace files, thus making source code control a reality.

The static analyses performed by the current version of ISUS include:

- Auditing the control flow of the module being changed by using directed graph techniques
- Modelling the calling hierarchy of the program's modules and globals

by analyzing subroutine and function calls and INCLUDE statements

- o Tracing a variable's usage within modules, and across module boundaries by analyzing parameter passing

These analyses and some reports based on them are triggered automatically by changes made to the program by the user. Other reports are available to the user on demand.

3. Demonstration Scenario

The demonstration will consist of one of our staff using ISUS to make changes to a program stored in a PM in order to demonstrate the functions and capabilities of the ISUS system. Tool Fair attendees will be able to suggest changes to be made, and will even be able to operate the system themselves, with prompting from our staff.

A single demonstration which demonstrates the technical capabilities of ISUS in a fairly complete manner should take 30 minutes or more, depending on the number of questions and suggestions offered by attendees. Hard copy of a sample ISUS session will be made available at the demonstration.

4. ISUS References

The following documents make up the reference library for ISUS. Documents marked with an asterisk (*) will be available at the Tool Fair.

E. F. Miller, Jr. and J. S. Praninskas, "Semantic Update Systems -- A Conceptual Analysis," RP-104, Software Research Associates, October 1977.

M. A. Hirschberg, W. G. Frickel, and E. F. Miller, Jr., "A Semantic Understanding System for Software Maintenance," Proc. 1979 Compcon Spring, March 1979.

* E. Sprinsock, "Examples of ISUS Use", TN-749/2, Software Research Associates, February 1981.

E. Sprinsock, "ISUS Command Reference Manual," RM-662/5, Software Research Associates, February 1981.

E. Sprinsock, "ISUS User Guide," RM-661/5, Software Research Associates, February 1981.

* "ISUS Fact Sheet", Software Research Associates, February 1981.

* "ISUS Implemented Command Summary," Software Research Associates, February 1981.

5. Demonstration Location and Time

Station 9, Wednesday, March 11, from 9:00 am until 4:00 pm, at specific

times to be posted at the station (4 - 6 demonstrations throughout the day).

6. The Demonstrators

Mr. MORTON A. HIRSCHBERG has been a Mathematician at the U. S. Army Ballistic Research Laboratory, Aberdeen Proving Ground, Maryand since 1973. His technical interests include software engineering management, software testing technology, and database design and management. In addition to his position with the Army, Mr. Hirschberg is a consultant to the Shock Trauma Unit at the University of Maryland's hospital complex in Baltimore, Maryland.

Previously, Mr. Hirschberg was a member of the technical staff at General Research Corporation, Santa Barbara, California, and Senior Computer Engineer at North American Aviation.

Mr. Hirschberg received his B. A. in Mathematics from the University of California, Los Angeles, and the M. A. in Psychology fro the University of California, Santa Barbara. During the 1973 academic year Mr. Hirschberg was Associate Professor of Psychology at UCSB. Mr. Hirschberg is a member of the ACM. Support for development of some versions of the ISUS system was provided under U. S. Army Contract No. DAAD05-78-C-1070.

Dr. EDWARD F. MILLER, JR., is Technical Director of Software Research Associates, San Francisco, California, a firm devoted to advanced computer technology and software applications. His interests include software engineering management, software testing technology, software maintenance technology, automated tool design and computer architecture.

Dr. Miller was previously Director of the Software Technology Center, Science Applications, Inc., San Francisco, and Director of the Program Validation Project at General Research Corporation, Santa Barbara, California. He received a BSEE at Iowa State University in 1962, an M. S. in Applied Mathematics at the University of Colorado in 1964, and the Ph. D. at the University of Maryland in 1968 where he was an Instructor from 1964 to 1968.

Dr. Miller is a member of the IEEE Computer Society, the ACM, SIAM and several honorary societies. He currently serves on several technical committees and is an Associate Technical Editor of COMPUTER Magazine.

Mr. JAMES B. HENDERSON is a programmer who specializes in the development of advanced concept tools for modern Software Engineering. His current research interest center on the development of techniques for hierarchical decomposition of programs based on their directed graph structure.

Since joining SRA in June 1979 he has been involved in a number of projects that support various Company projects. He has been responsible for production of the baseline version of SRTRAN, the Company's structured preprocessor system for Structured FORTRAN programming.

Mr. Henderson has supported the Company quality assurance program by participating in the development of a COBOL testbed system for the UNIVAC

1100 system, and through development of special versions of the COBOL Instrumentation Subsystem also for the UNIVAC 1100 environment. He is currently responsible for quality control on the entire COBOL Instrumentation Subsystem (all machine environments).

He has also participated in other Company efforts, including the development of the Semantic Update System (ISUS) series of automated software maintenance tools, and various in-house administrative processing facilities.

7. ISUS Contact

For further information please contact:

Dr. Edward Miller
Technical Director
Software Research Associates
P. O. Box 2432
San Francisco, CA 94126
(415) 957-1441

Mr. Morton Hirschberg
U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, MD 21005
(301) 278-4271

8. Proprietary Statement

This system is proprietary to Software Research Associates. The demonstrations of the system will be made for the purpose of describing the technical capabilities of the system only. Extensive documentation of a proprietary nature is available but will not be distributed to attendees without appropriate levels of protection of proprietary rights. Information and material that is distributed freely at the demonstrations carries no such restrictions, however.

SAMPLE COMMAND SEQUENCES

Figure A:

```
COMMAND? FIND /MINO/
  2480        MINCLAS=MINO(MINCLAS,ICURCLS)

COMMAND? VIEW 5
  2430 C
  2440 C      SET UP FOR THE NEXT CHARACTER
  2450        IF(ITKLST.GE.ITKMAX)RETURN
  2460 20     INDEX=INDEX+1
  2470        MINCM1=MINCLAS
  2480        MINCLAS=MINO(MINCLAS,ICURCLS)
  2490        MAXCLAS=MAXO(MAXCLAS,ICURCLS)
  2500        IF(ICURCLS.LE.DIGIT.AND.INDEX.LE.80)GOTO 20
  2510        LS=INDEX
  2520        MINCLAS=ICURCLS
  2530        MAXCLAS=ICURCLS

COMMAND? UPDATE DEL 2460
 STATEMENT  2460 DELETED.
 *** DELETED STATEMENT HAD A LABEL.
 *** PROGRAM MAY NO LONGER BE SYNTACTICALLY CORRECT.
 *** DELETED STATEMENT WAS ASSIGNMENT TO A VARIABLE.
 *** THE FOLLOWING LIST IS OF THE STATEMENTS WHICH
 *** LOGICALLY FOLLOW THE DELETED STATEMENT AND CONTAIN A
 *** REFERENCE TO THE VARIABLE ASSIGNED TO IN THE DELETED
 *** STATEMENT.
 ***       2500
 ***       2510
 ***       2590
 STATEMENT  1070:
 MESSAGE: TARGET OF GOTO NOT FOUND.
 STATEMENT  1190:
 MESSAGE: TARGET OF GOTO NOT FOUND.
 STATEMENT  2500:
 MESSAGE: TARGET OF GOTO NOT FOUND.
 *** AS STATED IN THE ABOVE MESSAGE(S), THE LOGICAL
 *** STRUCTURE OF THE CURRENT MODULE IS FLAWED, AND SO THE
 *** DIRECTED GRAPH OF THE MODULE IS UNUSABLE. ANY
 *** CONSEQUENCE ANALYSIS THAT REQUIRES THE DIRECTED
 *** GRAPH WILL BE INHIBITED UNTIL THE LOGICAL STRUCTURE
 *** OF THE MODULE IS SOUND.

COMMAND? RESTORE 2460
  2460 20     INDEX=INDEX+1

COMMAND?
```

SAMPLE COMMAND SEQUENCES

Figure B:

```
COMMAND? MODULE=LEXFR2
 LOADING MODULE LEXFR2
 MODULE LOADED, 258 STATEMENTS READ.

COMMAND? F /ITKDIR/
   550       DIMENSION ISTMT(666),ITKDIR(2,200)

COMMAND? VIEW 4
   510 C
   520       INCLUDE "STDFIL.COM"
   530       INCLUDE "DEBUG.COM"
   540 C
   550       DIMENSION ISTMT(666),ITKDIR(2,200)
   560       INTEGER QTYPE
   570       LOGICAL LCOMP
   580 C
   590       DATA IOPAR/1H(/

COMMAND? DIMENSION DEL ITKDIR 1
   550       DIMENSION ISTMT(666),ITKDIR(200)
   830       ITKDIR(J)=0
 UNSUBSCRIPTED OCCURRENCE OF VARIABLE IN STATEMENT   1020, NO CHANGE MADE.
 WRITING SECONDARY COMMANDS FOR STMT  1020:
  1020       CALL SUBLX1(I-4,I-1,ITKLST,ITKDIR,ITKMAX,LS)

SECONDARY: MODULE=SUBLX1
 SAVING MODULE LEXFR2
 MODULE SAVED, 258 STATEMENTS WRITTEN.
 LOADING MODULE SUBLX1
 MODULE LOADED,  44 STATEMENTS READ.

SECONDARY: DIMENSION DEL %4 1
 UNSUBSCRIPTED OCCURRENCE OF VARIABLE IN STATEMENT     20, NO CHANGE MADE.
   240       DIMENSION ITKDIR(200)
   310       ITKDIR(ITKLST)=ISTART
   320       ITKDIR(ITKLST)=I-ISTART+1

SECONDARY: MODULE=LEXFR2
 SAVING MODULE SUBLX1
 MODULE SAVED,  44 STATEMENTS WRITTEN.
 LOADING MODULE LEXFR2
 MODULE LOADED, 258 STATEMENTS READ.

COMMAND?
```

SAMPLE COMMAND SEQUENCES

Figure C:

```
COMMAND? P 1420
  1420        ITYPE=IRECTP(K)

COMMAND? VIEW 4
  1380 C
  1390 C      SET ELEMENTS OF PRESENT KEYWORD
  1400 C
  1410        NROW=K
  1420        ITYPE=IRECTP(K)
  1430        IGRUP=IGRP(K)
  1440        INDNTY=INDTYP(K)
  1450        IPRNCK=IPARCK(K)
  1460        NSEG=NSGS(K)

COMMAND? SUB /K/K+1/ 1410 1460
  1380 C
  1390 C      SET ELEMENTS OF PRESENT KEYWORD
  1400 C
  1410        NROW=K+1
  1420        ITYPE=IRECTP(K+1)
  1430        IGRUP=IGRP(K+1)
  1440        INDNTY=INDTYP(K+1)
  1450        IPRNCK=IPARCK(K+1)
  1460        NSEG=NSGS(K+1)

COMMAND?
```

SAMPLE COMMAND SEQUENCES

Figure D:

```
COMMAND? VIEW 4
   760 C     RUN THROUGH ALL ENTRIES
   770       I=1
   780 310   JBSO(I)=JSTAT*(KLSO(I)+I)
   790       I=I+1
   800       IF(I.LE.300)GOTO 310
   810       IF(JBSO(I).GT.0)GOTO 320
   820       CALL ERROR(JBSO(I))
   830 320   CONTINUE
   840 C

COMMAND? MOVE 810 830 B 800

COMMAND? POS 800
   800       IF(JBSO(I).GT.0)GOTO 320

COMMAND? VIEW 4
   760 C     RUN THROUGH ALL ENTRIES
   770       I=1
   780 310   JBSO(I)=JSTAT*(KLSO(I)+I)
   790       I=I+1
   800       IF(JBSO(I).GT.0)GOTO 320
   810       CALL ERROR(JBSO(I))
   820 320   CONTINUE
   830       IF(I.LE.300)GOTO 310
   840 C

COMMAND?
```

SAMPLE COMMAND SEQUENCES

Figure E:

```
COMMAND? VIEW 6
   210        JPAR=1
   220        DO 600 I=IFRST+1,ITKLST-1
   230        IF(ITGET(I,ITOK,LEN).NE.1)GOTO 300
   240        IF(ISTOK(ITKLST,ITKDIR,I,1,ILPR).NE.1)GOTO 100
   250        JPAR=JPAR+1
   260        GOTO 200
   270 100    IF(ISTOK(ITKLST,ITKDIR,I,1,IRPR).NE.1)GOTO 200
   280        JPAR=JPAR-1
   290 200    CONTINUE
   300        IF(JPAR.EQ.0)GOTO 600
   310        ILST=I
   320        ITBLPR=1
   330 300    CONTINUE

COMMAND? DUP 250 A 300

COMMAND? POS 270
   270 100    IF(ISTOK(ITKLST,ITKDIR,I,1,IRPR).NE.1)GOTO 200

COMMAND? VIEW 6
   210        JPAR=1
   220        DO 600 I=IFRST+1,ITKLST-1
   230        IF(ITGET(I,ITOK,LEN).NE.1)GOTO 300
   240        IF(ISTOK(ITKLST,ITKDIR,I,1,ILPR).NE.1)GOTO 100
   250        JPAR=JPAR+1
   260        GOTO 200
   270 100    IF(ISTOK(ITKLST,ITKDIR,I,1,IRPR).NE.1)GOTO 200
   280        JPAR=JPAR-1
   290 200    CONTINUE
   300        IF(JPAR.EQ.0)GOTO 600
   310        JPAR=JPAR+1
   320        ILST=I
   330        ITBLPR=1

COMMAND?
```

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . CODE INPUT
. . . SRTRAN
. . . FORTRAN
FUNCTION
. TRANSFORMATION
. . EDITING
. STATIC ANALYSIS
. . MANAGEMENT
. . . CONFIGURATION MANAGEMENT
. . . CHANGE CONTROL
. . ERROR CHECKING
. . DATA FLOW ANALYSIS
. . STRUCTURE CHECKING
OUTPUT
. USER OUTPUT
. . LISTINGS
. . DIAGNOSTICS
. MACHINE OUTPUT
. . SOURCE CODE OUTPUT
. . . SRTRAN
. . . FORTRAN

IMPLEMENTATION LANGUAGE: SRTRAN

TOOL PORTABLE: YES, TOOL SIZE: APPROX. 40K NC'SS

TOOL AVAILABLE: YES, PUBLIC DOMAIN: NO

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): LISCENSED SOFTWARE SYSTEMS

TOOL SUPPORTED: YES, TOOL SUPPORT: SOFTWARE RESEARCH ASSOCIATES

CONTACT: EDWARD F. MILLER, SOFTWARE RESEARCH ASSOCIATES, P.O. BOX 2432, SAN FRANCISCO, CA, 94126, USA, 415-957-1441
MORTON HIRSCHBERG, US ARMY BALLISTIC RESEARCH LABORATORY, ABERDEEN PROVING GROUND, ABERDEEN, MD, 21005, USA, 301-278-4271

A Software Engineering Tool Demonstration of

FAME

Front-End Analysis and Modeling Environment

A Product Of

HIGHER ORDER SOFTWARE, INC.

INTRODUCTION

FAME, the Higher Order Software, Inc. Front-End Analysis and Modeling Environment, is an interactive computer aided design tool that allows users to build, analyze, validate, store and graphically display models of systems. Use of FAME promotes higher productivity in the development of systems because it is easy to learn, and allows a spectrum of users to build many types of models necessary for system life cycle development and management, and insures consistency between them.

The techniques employed by HOS, Inc. have been developed over a number of years with a view toward providing a complete methodology for specification of complex, large scale systems. It has effectively been used for a variety of applications ranging in size from small and simple to large real-time systems. As part of its long range commitment to development of an integrated set of automated techniques that support the HOS methodology, HOS, Inc. has developed and is now marketing its automated modeling system.

Currently there are a number of manual methods in use that aid in design of systems. In general, these methods are costly to update and difficult to validate because they are maintained manually. Other systems exist to allow the user to describe the model in a computer so that it can be analyzed. These systems generally reside on large scale computers, are costly to operate and are not user friendly.

FAME is a computer-based system for interactively developing, analyzing and displaying HOS system models in a user friendly environment. The nature of the model is such, that when completed it can be the basis for projection to a variety of forms such as Structured Design Diagrams, Petri-Nets, Data Flow Diagrams, PSL/PSA, various HOL source codes, etc. The user's interface with the analyzer is easily recognized by any current user of a structured modeling approach; therefore extensive training is unnecessary. Furthermore, when all the system capabilities are used one can check on proper usage of Data Types, Functions and Control Structures and thereby add a new dimension to the design process that will lead to better, and more easily verified software designs.

SUMMARY OF FAME FEATURES

The Fame System features:

Prompted interactive development of models in a control map format which has been shown to be compatible with a wide variety of system modeling techniques.

Analysis of modeling errors based on a set of rules specified by the Higher Order Software methodology. Many common errors are found as the data is entered. The remainder can be found and corrected immediately after data entry.

Automated documentation of models in the form of printed specifications, graphic representations that include hierarchy charts and control maps. Projections to a variety of familiar forms are easily developed.

Software that is designed to operate on a spectrum of computer systems including single and multi-user computers with floppy disk to large scale midi-computer systems such as VAX 11/780. It is highly modularized and written in PASCAL with a view toward maximizing transportability.

System model development in a prompted alpha-numeric mode using either a standard screen-oriented terminal or a printing terminal.

SCENARIO FOR DEMONSTRATION

The Tools Fair demonstration will be conducted in three phases.

I- Brief Overview of Higher Order Software and FAME - The overview, confined to 15 minutes, will cover basics of HOS, their implementation in FAME and the results of FAME usage.

II- A live demonstration of the system will then be performed. Using a large screen video terminal and keyboard. Hard copy output will also be available. (15-30 minutes)

III- User questions and specific demo requests will be handled after the demo. At that time individuals can ask and get answers to any questions they have relative to the system. (30 minutes) Hands on or one-to-one demonstrations can be arranged by contacting Mr. Jack Rosenbaum at the conference.

FAME/HOS RELATED LITERATURE

The following literature is available at the conference.

- M. Hamilton and S. Zeldin, "The Relationship Between Design and Verification," The Journal of Systems and Software, 1, 29-56 (1979).
- J. Rosenbaum, W.R. Hackler, "Requirement Specifications for Embedded Astronautic Systems",Presented at the American Astronautical Society, October 1980.
- FAME System Description
- A set of Abstracts of other HOS literature that are available on request from Higher Order Software.

DEMONSTRATION SCHEDULE

Station 11, Wednesday, March 11
Starting 9 AM, 10 AM, 11 AM, 2 PM, 3 PM, 4 PM, 5 PM, 6 PM

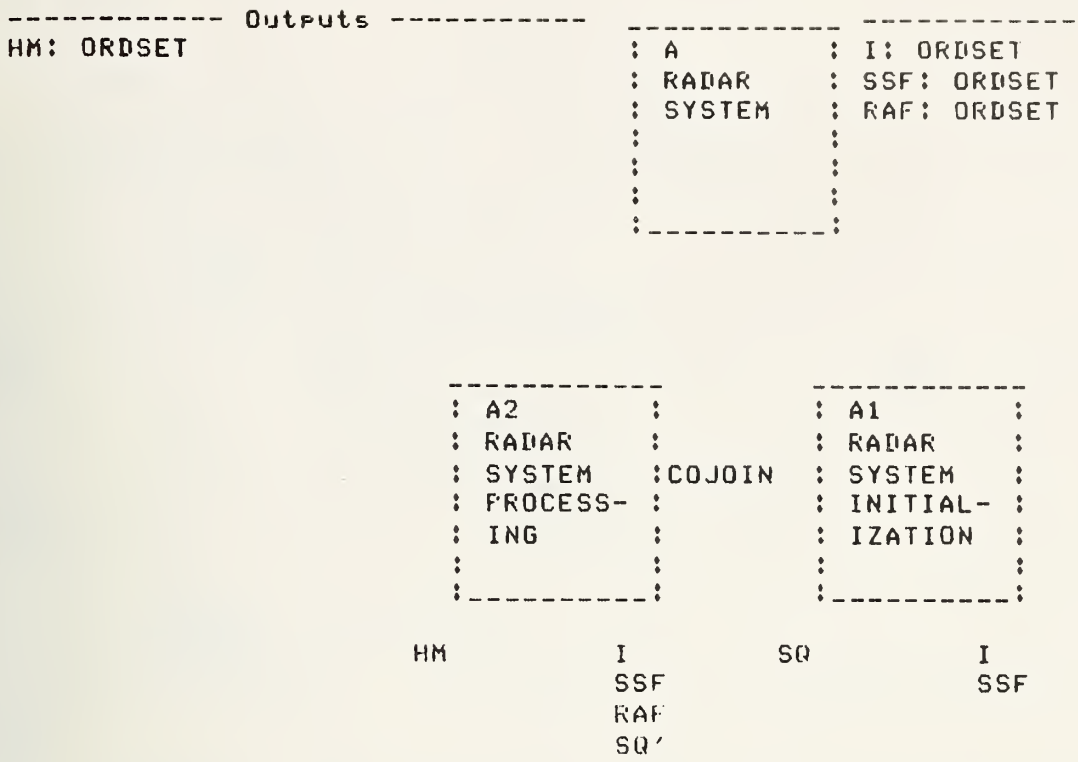
DEMONSTRATORS

Jacob D. Rosenbaum

Jacob D. Rosenbaum is the Director of the New York Office of HOS, Inc. He has 17 years aerospace experience in CAD/CAM Systems and Software Engineering. Mr. Rosenbaum has spoken frequently on various aspects of Software Engineering and is knowledgeable in the area of Software Tools. He was the principal engineer on the design of FAME and is responsible for developing related documenting and training courses.

Christopher Early

Christopher Early is a Programmer/Analyst at the New York Office of HOS, Inc. He has had experience with a wide variety of computer systems and programming languages. Mr. Early was a major contributor to the design and programming of FAME.



Keyname	Longname	Datatype	Part of
I	INFORMATION FILE		ORDSET
SSF	SEARCH SCAN FILE		ORDSET
RAF	RADAR ACTIVITIES FILE		ORDSET
HM	HANDOVER MESSAGE		ORDSET
SQ	ORDERED SET OF REQUESTS		ORDSET

WARNING: keyname 'SQ'' not found in global or local model.

TYPICAL PARENT/OFFSPRING DIAGRAM

TYPICAL HIERARCHICAL TREE DIAGRAM

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . VHLL INPUT
FUNCTION
. STATIC ANALYSIS
. . ERROR CHECKING
OUTPUT
. USER OUTPUT
. . DIAGNOSTICS
. . GRAPHICS
. . . HIERARCHICAL TREE
. . . CONTROL MAP

IMPLEMENTATION LANGUAGE: PASCAL

TOOL PORTABLE: NO, TOOL SIZE: MIN VIRTUAL MEMORY - 256K

COMPUTER (OTHER HARDWARE): VAX 11/780, CDC CYBER, IBM

OS (OTHER SOFTWARE): VM/CMS, NOS, VMS

TOOL AVAILABLE: YES, PUBLIC DOMAIN: NO

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): OBJECT CODE AVAILABLE UNDER LICENSE, TIME SHARED USAGE UNDER CONTRACT

TOOL SUPPORTED: YES, TOOL SUPPORT: HIGHER ORDER SOFTWARE

CONTACT: JACK ROSENBAUM, HIGHER ORDER SOFTWARE, INC., 131 JERICHO TURNPIKE, JERICHO, NY, 11753, USA, 516-997-7825

SCHEMACODE

AN INTERACTIVE SCHEMATIC PSEUDOCODE FOR PROGRAM DEVELOPMENT, DOCUMENTATION AND STRUCTURED CODING

(I2.I1.C2/T1.T2.T5.T6.S7.S11.S16./U3.U4.U5.M5.M6)

Pierre N. Robillard and Réjean Plamondon
École Polytechnique de Montréal

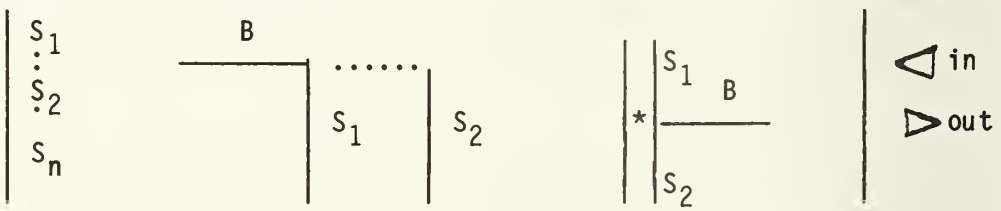
INTRODUCTION

A research team from the man-computer interface group at École Polytechnique de Montréal headed by Professors R. PLAMONDON and P.N. ROBILLARD has been working for the past three years on an innovative computer programming tool. The key elements of the methodology are a word-graphic type of communication called Schematic Pseudocode. Schemacode is the name of the tool that implements the schematic pseudocode. Schemacode is more than a comprehensive automatic documentation and coding tool, it links together the supervisors and the programmers on the development of a software project.

SCHEMATIC PSEUDOCODE: A Methodology

Schematic pseudocode (SPC) is a chart form which offers many advantages of the "alphanumerical" pseudocode (executable, self-explanatory...) and all of the advantages of pictorial structured algorithm. Each program can be graphically expressed in terms of actions which can be:

1) Sequential 2) Conditional 3) Repetitive 4) I/O



In this description, we use S for statement and B for Boolean expressions. The graphic symbolism is exclusively designed to represent the structure of the process while words are used to describe statements and Boolean expressions.

The problem of solving all equations of the form $ax^2 + bx + c = 0$ for several given sets of a, b, c values, shown below, is a simple example that illustrates the use of schematic pseudocode. Complex problem solving using SPC will be demonstrated.

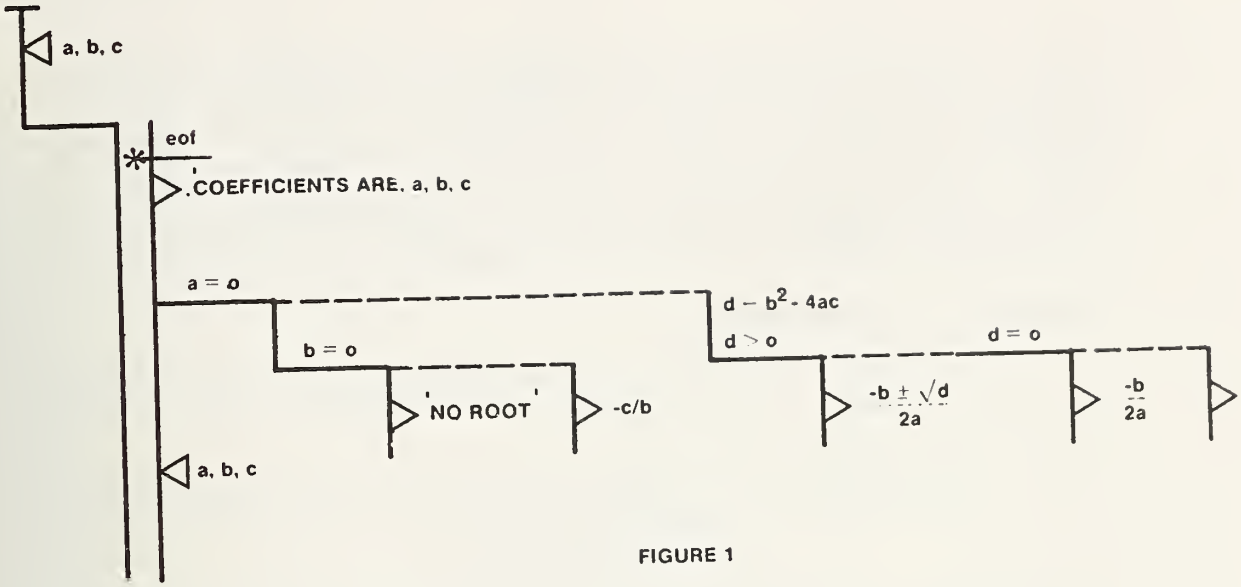


FIGURE 1

SPC is an easy-to-learn language and its unique property of word-graphic type of communication can satisfy the needs of both the supervisor and the programmer.

SCHEMACODE: A Tool

Schemacode is a software package running on an IBM 360. The terminal (VT-100 compatible) makes it interactive. The primary task of Schemacode is to assist users in the development, documentation and structured coding of programs. It usually transmits the source program to the main computer for execution. In the development phase of a project the SPC is output at the graphic printer, while in the coding phase, the structured listing can be output.

Once the need or the task to be performed by software is correctly defined, the system analyst enters the main control structures at the terminal. This first step is called refinement #0. The first or any subsequent refinement can contain up to 10 structures. This limit is not restrictive. It rather appears as a way to force a top-down approach. Schemacode asks the user to identify each of the segments. The identifications expressed in natural language will be automatically integrated later as comments in the formal language program. The user can proceed one step down by refining one segment. Schemacode will ask for identification of every newly defined segment or control structure.

The process can go as far down as required. Once the desired level is reached, Schemacode will automatically integrate all the steps and provide a complete chart of the process. The user can recall any refinement, redraw it or modify it. All modifications to structures or comments will be automatically integrated into the whole. When the process is pursued to the code level a structured listing is provided in a selected language. A real advantage provided by Schemacode is that every program developed has a unique up-to-date documentation listing.

SCHEMACODE: The Demonstration

A typical scenario of Schemacode demonstration will include the following steps:

1. What Schemacode can do for you?
   - automation of:
     - step-wise refinement and top-down approach
     - documentation
     - structured coding

2. An overview of schematic pseudocode (SPC).

3. Ready for an example?
   - design of a program with schemacode (data processing, engineering)
   - modification of an existing non documented and unstructured Fortran program.

STATION, DAY AND TIME

Station 1, Thursday, March 12 from 10:00 am until 6:00 pm. The demonstration will take about thirty minutes, according to the posted schedule. Schemacode literature (reference manual, user's guide, collected examples...) will be available on request.

THE DEMONSTRATORS

Dr. Réjean PLAMONDON and Dr. Pierre N. ROBILLARD are respectively assistant professor of electronics and associate professor of computer science at Ecole Polytechnique at the Université de Montréal, CANADA. They lead a research group currently involved in projects based on man-computer interaction.

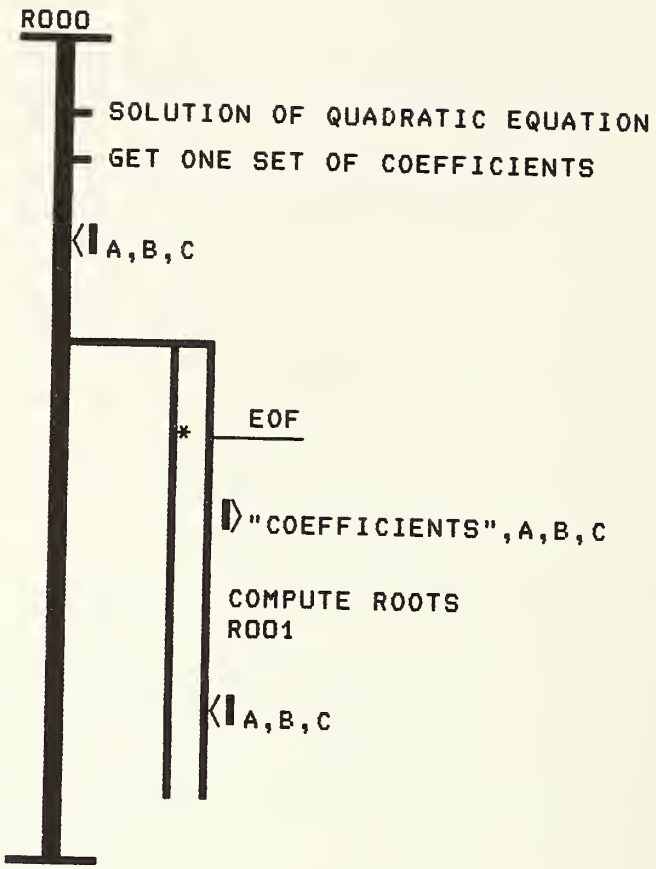
ACKNOWLEDGEMENTS

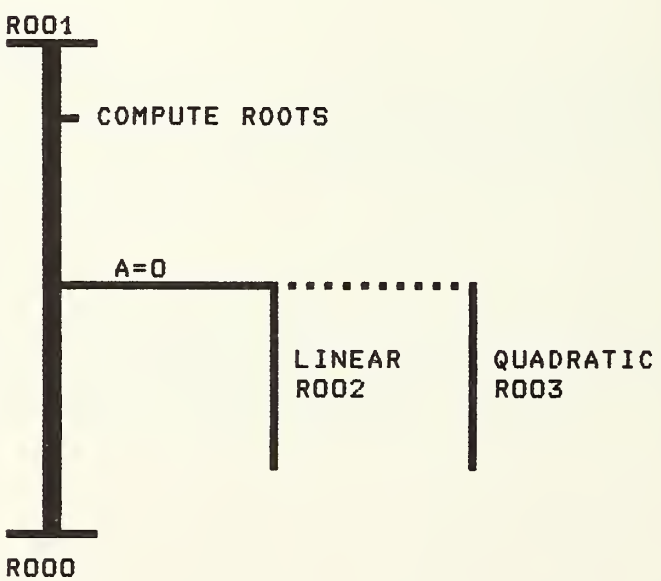
This research is partially supported by CRSNG grant RD-60, FCAC grant EQ-1727. The authors wish to thank everyone who has contributed to the development of the tool, in particular Mr. Augustin BRAIS, Mr. Tan Phalkun and professor Daniel THALMANN.

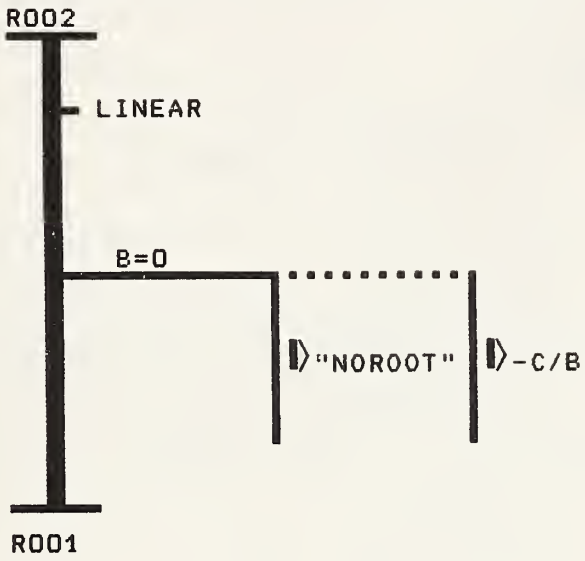
EXAMPLE OUTPUT

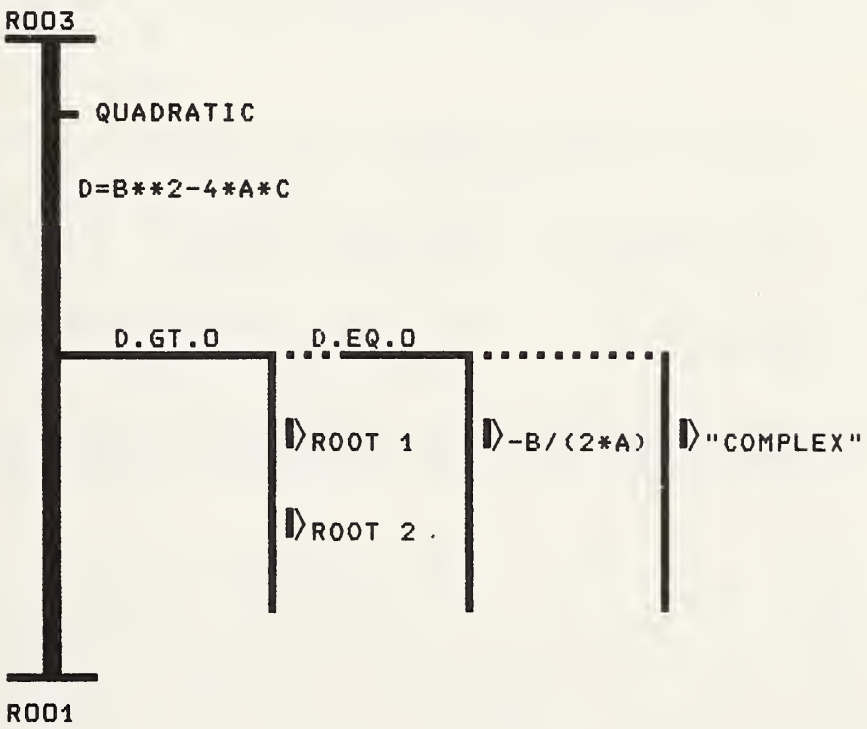
A program for solving the quadratic equation as derived with the help of SCHEMACODE is illustrated in the following pages.

The first step R000 is to get one set of input parameters which contains three values, one each for a, b, c. Since this problem requires the solution of the equation for several sets of a, b, c values the second step R001 is to loop over all the remining data (EOF). Before the looping is resumed, the roots must be computed from the data just read. There are two alternatives from which we must select, the cases R002 which have less than two roots and the cases R003 which have two roots (a = 0). The former concerns solving the equation bx + c = 0 which may have either a single root or no roots at all. For the latter it is known that two roots exist, but whether they are real or complex must still be decided.


R000
SOLUTION OF QUADRATIC EQUATION
GET ONE SET OF COEFFICIENTS
A,B,C
*
EOF
"COEFFICIENTS",A,B,C
COMPUTE ROOTS
R001
A,B,C


R001
COMPUTE ROOTS
A=0
LINEAR
R002
QUADRATIC
R003
R000


R002
LINEAR
B=0
"NOROOT"
-C/B
R001


R003
QUADRATIC
D=B**2-4*A*C
D.GT.0
D.EQ.0
ROOT 1
ROOT 2
-B/(2*A)
"COMPLEX"
R001

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . VHLL INPUT
. . . SCHEMATIC PSEUDOCODE
. . TEXT INPUT
FUNCTION
. TRANSFORMATION
. . EDITING
. . FORMATTING
. . RESTRUCTURING
. STATIC ANALYSIS
. . MANAGEMENT
. . COMPLEXITY MEASUREMENT
OUTPUT
. USER OUTPUT
. . LISTINGS
. MACHINE OUTPUT
. . SOURCE CODE OUTPUT
. . . FORTRAN

IMPLEMENTATION LANGUAGE: FORTRAN

TOOL SIZE: 150 K

COMPUTER (OTHER HARDWARE): IBM 360/370

TOOL AVAILABLE: YES, PUBLIC DOMAIN: NO

TOOL SUPPORTED: YES, TOOL SUPPORT: ECOLE POLYTECHNIQUE DE MONTREAL

CONTACT: PIERRE N. ROBILLARD, DEPT OF EE, ECOLE POLYTECHNIQUE, BOX 6079, STATION A, MONTREAL, 83C3A7, CANADA, 514-344-4711

INSTRU:
AN AUTOMATED SOFTWARE INSTRUMENTATION SYSTEM

A Software Engineering Tool Demonstration

J. C. Huang
University of Houston

1. INTRODUCTION

This proposal is submitted in response to the call for software engineering tool developers to demonstrate their accomplishments at the Fifth International Conference on Software Engineering in San Diego, March 9-12, 1981.

We will be demonstrating the INSTRU software system currently developed by the Program Testing Project at the University of Houston.

2. SUMMARY

INSTRU is an experimental software tool that can be used to increase the error-detection capability of a program test by instrumenting the program for data-flow anomaly detection and symbolic-trace generation. The ideas involved are briefly described below:

(a) Data-Flow Anomaly

In execution a program may act on a variable in three different ways: viz., define, reference, and undefine. A variable is defined in a statement if an execution of the statement assigns a value to that variable. A variable is referenced in a statement if an execution of the statement requires that the value of that variable be obtained from the memory. Thus, in the assignment statement

```
x = x + y - z
```

y and z are both referenced, whereas x is first referenced and then defined. A variable may become undefined in many circumsstances. For example, in a Fortran program, the index variable of a DO statement becomes undefined when the loop is terminated, and the local variables of a subprogram become undefined when RETURN statement is executed.

While a program is being executed, a sequence of action will be taken on each variable in the program. The design of a programming language is such that a variable cannot be referenced unless its value is previously defined. Furthermore, there is no need to define a variable unless it is to be referenced (i.e., its value is to be used) later. Thus, if we find that a variable in a program is (1) undefined and referenced, (2) defined and then undefined, or (3) defined and then defined again, we may then reasonably conclude

that a programming error might have been committed.

The three types of data flow anomalies mentioned above can be detected by instrumenting the program to be tested as described in References [1-3], and then execute it for a properly chosen set of test cases.

(b) Symbolic Trace

A symbolic trace [4] is a linear listing of source statements and branch predicates that occur along an execution path in a program. Essentially, it represents the sequence of program components exeamined by a programmer in the process of code walk-through, or in performing symbolic execution. As such, it can be utilized to facilitate program debugging and verification.

A symbolic trace provides us with three types of information that are particularly useful in connection with program testing: (1) it explicitly describes the path along which the program is executed, (2) it displays the conditions that must be satisfied at various points along the path, and (3) it clearly describes the computation performed in terms of the statements executed. The first type of information can be used to determine the extent of test coverage. If the coverage need to be increased, the second type of information can be used to select additional test cases. If a test failed, the third type of information can be used to facilitate location of programming errors. The presence of a programming error is generally more obvious on a symbolic trace than on the program text.

3. A SCENARIO OF AN INSTRU DEMONSTRATION

Several example programs will be used to demonstrate how the software tool INSTRU works. The attendees can also try it out by using a sample program of their choice as follows:

(1) Enter the program and save it in a file. The program should be written in ANSI Fortran and meet certain minor syntactic constraints listed in the user's guide [5].

(2) Instrument the program for data-flow anomaly detection by invoking INSTRU.D subsystem.

(3) Execute the instrumented program for a number of test cases interactively on the terminal. If there is a data-flow anomaly on the execution path, a message will appear on the terminal indicating the nature and location of the anomaly as well as the variable involved.

(4) Instrument the program for symbolic-trace generation by invoking INSTRU.S subsystem.

(5) Execute the instrumented program interactively for the same test cases used in step (3). The symbolic trace that describes the test path will be generated in the process.

Remarks: If the test failed, or if a data-flow anomaly was detected in step (3), one can examine the symbolic trace to find the source of error. Ususally the presence of a programming error is more obvious on a symbolic trace than on a program text. It is useful to study the symbolic trace even if the test was successful. One may discover that the test result is fortuitously correct.

4. REFERENCES

[1] J. C. Huang, "Program Instrumentation and Software Testing," COMPUTER, vol. 11, no. 4, April 1978.
[2] J. C. Huang, "Program Instrumentation: a Tool for Software Testing," INFOTECH State of the Art Report: Software Testing, Volume 2, 1979.
[3] J. C. Huang, "Detection of Data Flow Anomaly through Program Instrumentation," IEEE Transactions on Software Engineering, vol. SE-5, no. 3, May 1979.
[4] J. C. Huang, "Instrumenting Programs for Symbolic-Trace Generation," COMPUTER, vol. 13, no. 12, December 1980.
[5] INSTRU User's Guide, Department of Computer Science, Univ. of Houston. (will be made available at the Tool Fair)

5. STATION, DAY, AND TIME

Station 2, Thursday, March 12, from 10:00 am to 2:00 pm.

6. THE DEMONSTRATOR

J. C. Huang:

J. C. Huang is a professor of computer science at the University of Houston. He has authored numerous articles on program analysis and testing, and is the principal investigator of a research project on program testing currently supported by National Science Foundation. He received his MS degree from Kansas State University in 1962, and his Ph.D. degree from the University of Pennsylvania in 1969.

address: Department of Computer Scince
University of Houston
Houston, TX 77004
telephone: 713 749-2856/4791

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . CODE INPUT
. . . FORTRAN
FUNCTION
. TRANSFORMATION
. . INSTRUMENTATION
. DYNAMIC ANALYSIS
. . TRACING
. . . DATA FLOW
. . . LOGIC FLOW
. . . PATH FLOW
OUTPUT
. USER OUTPUT
. . TABLES
. . LISTINGS
. MACHINE OUTPUT
. . SOURCE CODE OUTPUT
. . . FORTRAN

IMPLEMENTATION LANGUAGE: FORTRAN IV

TOOL PORTABLE: NO

COMPUTER (OTHER HARDWARE): HONEYWELL 6000

OS (OTHER SOFTWARE): 4JS2

TOOL AVAILABLE: YES, PUBLIC DOMAIN: YES

TOOL SUPPORTED: NO

CONTACT: J. C. HUANG, UNIVERSITY OF HOUSTON, CENTRAL CAMPUS, DEPT OF COMP SCI, HOUSTON, TEXAS, 77004, USA, 713-749-2856

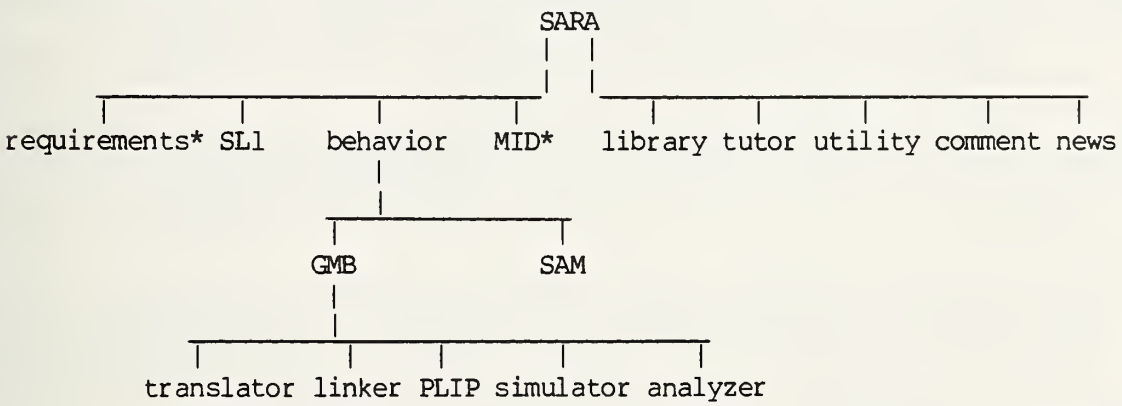
Control-Flow Analysis in SARA +

Rami R. Razouk
Computer Science Department
University of California, Los Angeles

1. Introduction:

SARA (System ARchitects' Apprentice) is a computer-aided design system, currently under development at UCLA, which supports a structured multi-level design methodology for the design of hardware or software systems. It comprises a number of language processors and tools for assisting designers using the SARA methodology, together with a user-interface capability for assisting designers using the SARA system. The SARA system is implemented on the MIT Multics system and is readily accessible through ARPANET or TELENET.

The hierarchy of the SARA system is illustrated below:



The leaves of the left branch of the tree are the tools supporting the methodology while the leaves of the right branch are the tools supporting the SARA system [FenR80]. The asterisks indicate that the tools have not been implemented yet.

2. Summary:

The SARA methodology is requirement driven and it supports both top-down and bottom-up design procedures. Each step of a design starts with the definition of the requirements for the system and the assumptions about the system's environment. The tools to accept and analyze requirement definitions are currently under development and are discussed in a recently completed dissertation [WinJ80].

+ This work was supported by the Department of Energy, Contract No. DE-AF03-765F0034 P.A., No. DE-AT-036, ER70214, Mod. A006.

Hardware and software systems are designed in SARA by modelling their structure and behavior. The tools SL1 (Structural Language 1) and GMB (Graph Model of Behavior) support definition of structural and behavioral models respectively. GMB allows definition of behavior in three domains: control-flow, data-flow and interpretation. The GMB Translator is the language processor for the control and data-graph definitions; PLIP is the language processor for the interpretation.

GMB models are mapped to structures and are denoted SL1-GMB models. The linking of GMBs mapped to connected structures is processed by the GMB-Linker.

The GMB Simulator provides an interactive simulation environment which permits experiments on the behavioral models. The GMB Analyzer allows designers to perform formal analysis on the GMB control-graph [RazR80a,b].

Behavioral Attributes can be associated with structures by means of SARA's Attribute-Based Model (SAM) [SamA81].

A top-down design strategy can be applied by refining structural and behavioral models of the system. A bottom-up design strategy can be applied by composing structural and behavioral models of existing building-blocks [DroJ80].

In software design, a path between modelling and code is provided by the definition of the structure of code and a mapping between the structure of the models and the code structure. The MID (Module Interface Description) tool (not yet implemented) permits these definitions [PenM80,81].

3. Scenario:

SARA's Control-Flow Analyzer provides the capability for formally analyzing those aspects of behavior modelled explicitly in the control domain of the GMB. Typically such behavior includes flow-of-control, process synchronization and resource sharing. The analyzer is capable of performing an exhaustive analysis to detect all potential deadlocks and cycles. The results of the analysis are presented to designers in a very readable form which highlights the potential areas of difficulty and provides clues as to the possible sources of the difficulties. The Control-Flow Analyzer combats the "state explosion" problem by providing a mechanism for performing "proper" abstraction using the "reduction procedure" developed at UCLA.

In this demonstration methods for modelling behavior in the control domain will be presented. The analysis techniques, including reduction, will be explained through the use of some simple examples. The full power of the analysis machinery will be demonstrated by modeling and analyzing a complex CCITT communication protocol standard: X.21.

Attendees will be permitted to enter models in the form of GMBs or Petri Nets and to exercise all of the analysis mechanisms provided.

4. SARA Literature:

[DroJ80] Drobman, J. "Building Block Modeling Methodology for Composition of Microprocessors-Based Digital Systems," Ph.D. Dissertation, Computer Science Department, University of California, Los Angeles, July 1980.

[EstG78] Estrin, G. "A Methodology for design of digital systems - supported by SARA at the age of one," AFIPS, Proceedings of the National Computer Conference, June 1978.

[FenR80] Fenchel, R.S "Interactive Systems with Integral Help," Ph.D. Dissertation, Computer Science Department, University of California, Los Angeles, July 1980.

[PenM80] Penedo, M.H. "The Use of a Module Interface Description in the Synthesis of Reliable Software Systems," Ph.D. Dissertation, Computer Science Department, University of California, Los Angeles, November 1980.

[PenM81] Penedo, M.H., et. al. "An Algorithm to support Code-Skeleton Generation for Concurrent Systems," Proceedings of the 5th International Conference on Software Engineering, San Diego, California, March 1981.

[RazR80a] Razouk, R. "Computer-Aided Design and Evaluation of Digital Computer Systems," Ph.D. Dissertation, Computer Science Department, University of California, Los Angeles, August 1980.

[RazR80b] Razouk, R. and G. Estrin "Modeling and Verification of Communication Protocols in SARA: The X.21 Interface," IEEE Transactions on Computers, December 1980.

[SamA81] Sampaio, A.B.C. "A Scheme of Attributes for Checking Design Inconsistencies," Ph. D. Dissertation, Computer Science Department, University of California, Los Angeles, to be completed 1981.

[WinJ80] Winchester, J. "Requirements Definition and its Interface to the SARA Design Methodology for Computer-Based Systems," Ph.D. Dissertation, Computer Science Department, University of California, Los Angeles, November 1980.

5. Station, Day and Time:

Station 3, Thursday March 12, from 11:00 a.m. until 3:00 p.m.

6. The Demonstrator: Dr. Rami R. Razouk

Dr. Razouk is currently an Assitant Professor in Residence in the Computer Science Department at UCLA. His research interests include computer-aided design and evaluation of digital computer systems. He received his Ph.D. in Computer Science from UCLA where he led the development of the SARA system. He also holds an M.S in Computer Science and a B.S. in Enginerring from UCLA.

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . VHLL INPUT
. . . SL1
. . . GMB
. . . BNF
. . TEXT INPUT
FUNCTION
. TRANSFORMATION
. . TRANSLATION
. . FORMATTING
. . RESTRUCTURING
. STATIC ANALYSIS
. . DATA FLOW ANALYSIS
. . STRUCTURE CHECKING
. . CROSS REFERENCE
. . CONSISTENCY CHECKING
. . COMPLETENESS CHECKING
. . SCANNING
. DYNAMIC ANALYSIS
. . SIMULATION
OUTPUT
. USER OUTPUT
. . GRAPHICS
. . LISTINGS
. . USER-ORIENTED TEXT
. . . DOCUMENTATION
. . . ON-LINE ASSISTANCE
. MACHINE OUTPUT
. . SOURCE CODE OUTPUT
. . DATA OUTPUT
. . PROMPTS

IMPLEMENTATION LANGUAGE: PL/1, TOOL PORTABLE: NO, TOOL SIZE: 25000 LINES OF PL/1 SOURCE

COMPUTER (OTHER HARDWARE): HONEYWELL, OS (OTHER SOFTWARE): MULTICS

TOOL AVAILABLE: YES, PUBLIC DOMAIN: YES, TOOL SUPPORTED: NO

CONTACT: G. ESTRIN, UNIVERSITY OF CALIFORNIA, COMPUTER SCIENCE DEP, BOELTER HALL 3732, LOS ANGELES, CA, 90024, USA, 213-825-8878

A User Interface for Online Assistance

Nathan Relles SPERRY UNIVAC
Lynne A. Price BNR, Inc.

1. INTRODUCTION

The tool we will demonstrate is described in a paper to be presented at the conference: "A User Interface for Online Assistance." The user interface enables programmers to provide and maintain online aids in an interactive system. Through the interface, users are given a set of consistent and unobtrusive aids that display summary information, command descriptions, explanations of error messages, and other online documentation. The interface will be demonstrated from the views of both the end-user and the programmer. It has been implemented on a Sperry Univac 1100 and on a PDP 11/70 running on the UNIX operating system.

2. SUMMARY

From the user's point of view, special assistance functions are always available to request information or to obtain successively more detailed explanations of a displayed message. These requests are entered through function keys or as special codes and do not affect the interpretation of other input. Different types of aids may be requested. For instance, a user who makes an error when responding to a system prompt can obtain further explanation of the original question, further explanation of the error message, or examples of correct responses.

From the programmer's point of view, each multi-level message (called a SCRIPT) is written as a separate file. In both implementations, all the scripts required for a given application are grouped together in a single file called the MESSAGE FILE, although single scripts can still be accessed independently by any text editor. To provide online assistance, a program must request all user input through the interface, which screens out assistance requests. When such requests are detected, appropriate help is provided and new input is solicited; other inputs are returned to the calling routine. Some examples of the types of aids that can be provided by the interface are shown on the following page. Of course, different aids are possible, depending on the nature of an application and the characteristics of its users. The assistance request codes shown here (?ERROR, ?QUESTION, etc.) are also intended only as examples; the programmer may define other codes or, if available, suitable function keys.

USER MAY ENTER	TO OBTAIN
?ERROR	successively more detailed explanations of a displayed error message
?QUESTION	successively more detailed explanations of a displayed question or prompt
?EXAMPLE	successive examples of correct input or valid commands
?DEFINE term	explanation or definition of a specified term
?FORMAT command	a description of the format of a specified command
?MENU	a list of allowable commands
?DOC section	a display of a specified section of documentation
?STATUS	a description of the current value of various system parameters
?INSTRUCT	instruction on the use of the system
?NEWS	news of interest to users of the system
?HELP	a list of available user aids

3. SCENARIO

Attendees will be able to test existing software that incorporates the user interface. They will be encouraged to ask for different types of online assistance and explanations of displayed messages. Different programs that provide the interface will be demonstrated and no prior experience with the programs will be required. Attendees will also be able to inspect the corresponding source code and the text of assistance messages as entered by the programmer.

4. LITERATURE DESCRIBING THE INTERFACE:

Relles, N., and Price, L. A. "A User Interface for Online Assistance" paper #102 to be presented at the Fifth International Conference on Software Engineering, 1981. This paper contains an extensive bibliography on Online Assistance.

Relles, N. The Design and Implementation of User–Oriented Systems, Doctoral Dissertation, University of Wisconsin–Madison, 1979; University Microfilms No. 79–24,190.

Price, L. A. Representing Text Structure for Automatic Processing, Doctoral Dissertation, University of Wisconsin–Madison, 1978; University Microfilms No. 78–15,065.

5. **STATION, DAY, AND TIME:**

Station 4; Thursday, March 12; from 1:00 pm to 7:00 pm.

6. **DEMONSTRATORS**

Nathan Relles is continuing his research on improving the ease with which computer systems can be learned and used. As a member of Sperry Univac's Software Research Deparment, his current research is in natural language access to databases and in online assistance. He received a Ph.D. in Computer Sciences from the University of Wisconsin, where he was employed by the Madison Academic Computing Center in developing and maintaining several database management systems. He also holds M.S. and B.A. degrees in Computer Sciences from Wisconsin. Prior to graduate school, he worked for Sperry Univac in the development of Computer Assisted Instruction systems.

Lynne A. Price is a member of the Computer Science Research Department of BNR, Inc. and is currently engaged in research on computer–controlled typesetting as well as online documentation, projects that reflect her interests in computational linguistics and user–oriented computing. She recently completed work begun as a dissertation in computer sciences at the University of Wisconsin in Madison and continued during a postdoctoral research associateship at NASA's Ames Research Center. Her association with co–author Nathan Relles stems from the LEXICO system on which they both worked during graduate school. Her undergraduate education was at the University of Washington in Seattle, where she received a B.S. in mathematics and an M.S. in computer science.

```
Tue Apr  7 12:48:33 1981
peruse V1.5 3/11/81

One moment, please.

For a list of available user aids, enter ?

P 0 S1-1  (Page 4 Line 1 - Page 23 Line 12)
Request? ?
Enter one of the following for help:
  ?error, ?menu , ?request KEYWORD, ?define TERM
For explanation of these help functions enter
  ?
again.
Request? ?
The following codes may be entered at any time (with no preceding
blanks) to obtain the indicated type of help:

?error (or ?e)      an explanation of the most recent error message.
                    ?e may be entered several times for progressively
                    more detailed explanations.

?menu (or ?m)       a list of allowable user inputs.  After a system
                    question, ?m may be entered several times for
                    progressively more detailed descriptions of the
                    desired information.

?request KEYWORD    a description of the indicated request (KEYWORD
 (or ?r KEYWORD)    is one of the request keywords listed by ?m)

?define TERM        a definition of the indicated term (if TERM is
 (or ?d TERM)       omitted a list of the terms for which definitions
                    are available is printed)
Request? ?menu
Any of the following requests may be entered:

text                        to display the text of P 0
contents                    to list the subdivisions of P 0
index                       to list the terms pertaining to P 0
status                      to show the status of P 0


about TERM1 TERM2 ...       to request information about the
                              specified combination of terms
passage N                   to display the requested passage
terms TERM -BEFORE AFTER    to list available terms
quit                        to end the session with peruse


For a description of one of the above requests, enter

      ?r KEYWORD

where KEYWORD is the first word in one of the requests above.
```

```
Request? ?define
Definitions are available for the following terms:

choice
combination
contents
index
passage
passage number
P number
peruse
request
status
term
text
Request?  ?define P number
"Peruse" operates on a text that has been divided  into  sections
called  "passages".   Each  passage  may  itself  be  subdivided.
Passages are  assigned  identification  numbers  (P  numbers)  to
correspond  to  line  numbers  in a complete outline of the text.
Thus, every passage's number is smaller than the numbers  of  its
subdivisions  and  also is smaller than the numbers of succeeding
sections.
Request? about call transfer
about call transfer
Passages:
1     P 18 Night Answer From Any Station  (Page 7 Lines 24 - 30)
2     P 38 Call Transfer  (Page 12 Line 1 - Page 13 Line 9)


Term Combinations:
3     key call transfer
4     call transfer lamp
5     call feature transfer
Choice?  (?menu for explanation)  ?menu
Enter  a  number  between  1 and 2 to inspect  one  of  the  passages
listed above.  Enter a number between 3 and 5 to inspect passages
relevant to one of the combinations of terms  listed  above.  Use
the "status" request to review the possibilities.

Any of the following requests may also be entered:


about TERM1 TERM2 TERM3 ...    to search for information relevant to
                                 the specified combination of terms
passage N                      to inspect the specified passage
terms TERM -BEFORE AFTER       to display terms
quit                           to terminate the session with "peruse"


For more information about a specific request, enter
     ?r KEYWORD
where KEYWORD is the first word in the request
```

```
Choice?  (?menu for explanation) 7
7: bad choice; enter a number between 1 and 5.
Choice?  (?menu for explanation) ?error
Enter a number between 1 and 2 to inspect  one  of  the  passages
listed above.  Enter a number between 3 and 5 to inspect passages
relevant to one of the combinations of terms  listed  above.  Use
the "status" request to review the possibilities.



Choice?  (?menu for explanation) ?error
No further explanation of  the  error  is  available.  For  more
information, enter
     ?
for a list of available user aids.  If you are unable to proceed,
contact Lynne Price (price on UNIX/C) at extension 2444.
Choice?  (?menu for explanation) 1
P 18 Night Answer From Any Station  (Page 7 Lines 24 - 30)
Request?  Any of the following requests may be entered:

text                          to display the text of P 18
contents                      to list the subdivisions of P 18
index                         to list the terms pertaining to P 18
status                        to show the status of P 18


about TERM1 TERM2 ...         to request information about the
                                specified combination of terms
passage N                     to display the requested passage
terms TERM -BEFORE AFTER      to list available terms
quit                          to end the session with peruse


For a description of one of the above requests, enter

      ?r KEYWORD

where KEYWORD is the first word in one of the requests above.
Request? text
P 18 Night Answer From Any Station  (Page 7 Lines 24 - 30)


Night Answer From Any Station

With this type of Night Service, incoming calls will  ring  on  a
night  bell(s)  and  can be answered from any telephone. When the
Night Bell rings, lift the receiver on any  phone  and  dial  the
Night  Service  code.  To  transfer  the  call to the appropriate
extension,  follow  the  "Call  Transfer"  or  the  "Conference"
procedure.
```

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . TEXT INPUT
FUNCTION
. STATIC ANALYSIS
. . MANAGEMENT
. . . DOCUMENTATION MANAGEMENT
OUTPUT
. USER OUTPUT
. . USER-ORIENTED TEXT
. . . DOCUMENTATION

IMPLEMENTATION LANGUAGE: FORTRAN, C

TOOL PORTABLE: YES

COMPUTER (OTHER HARDWARE): UNIVAC 1100, PDP 11/70

OS (OTHER SOFTWARE): EXEC 8, UNIX

TOOL AVAILABLE: YES

CONTACT: NATHAN RELLES, SPERRY UNIVAC, MS 2G3, PO BOX 500, BLUE BELL, PA, 19424, USA, 215-542-2387

FORTRAN 77 ANALYZER

Tool Demonstration For ICSE5
John Barkley, Patricia Powell

1. Introduction

The FORTRAN 77 Analyzer was developed by TRW under contract to the National Bureau of Standards. It provides static and dynamic analysis of a FORTRAN program. It is in the public domain and will be available through National Technical Information Service(NTIS), U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161 late summer or early fall of 1981. The Analyzer is written in FORTRAN 77 and is portable. It requires 64K of memory.

2. Summary

The FORTRAN 77 Analyzer evaluates the structure of FORTRAN 77 software both statically, based upon the source code, and dynamically, during execution. The system will produce reports describing the findings of the static and dynamic analysis requested by the user. The Analyzer system is composed of three parts: the pre-processor, the instrumented source program and the post-processor. The development of the FORTRAN 77 Analyzer was done by the TRW Defense and Space Systems Group for the National Bureau of Standards Institute for Computer Sciences and Technology. The FORTRAN 77 Analyzer is expected to contribute to the establishment of a set of generic tools and techniques for the development of quality software in the Federal Government.

The pre-processor accepts a FORTRAN 77 source program and options selected by the user to control the analysis process. The source code is divided into segments, a group of consecutively executable segments having only one entry and exit. Code is added to the program so dynamic execution statistics can be accumulated.

The user can ask the pre-processor for any or all of the following options:

- -Assertion Checking: An assertion is a special form of the comment statememt which contains a FORTRAN logical expression. If this option is requested, code is added to monitor the values of assertion expressions.
- -Tracing: Code is added to print segment numbers during program execution.
- -History file management: Execution statistics are accumulated over several runs

of the instrumented source program.

The output of the pre-processor consists of:

- -the instrumented source program
- -an annotated listing of the source program highlighting segments, assertions and statements failing syntax analysis
- -a subroutine call tree listing
- -a static distribution of statement types.

The second part of the analyzer system is the instrumented source program. The instrumented source program accepts the user program's normal input and produces the user program's normal output. In addition, it can request segment trace ranges and input a history file, if these options were selected in the pre-processor. The outputs of the instrumented source program are:

- -a listing of the dynamic execution frequencies of segments
- -a listing of the segments executed within the trace ranges selected
- -a listing of the assertion identifiers whose values were false
- -an updated history file.

The final part of the analyzer system is the post-processor. It takes information from the annotated listing, the summary of statement types(which were created by the pre-processor) and the history file which was produced by the instrumented source program. It produces the following reports under user control:

- -program and module names
- -number of statements, comments and syntax errors
- -invocation frequencies for each routine
- -percentage of statement and segment coverage
- -dynamic frequencies of segments
- -static and dynamic frequencies by statement types
- -source program listings annotated with segment usage and assertion frequencies.

3. Demonstration Scenario

A program, with errors, will be run using the Analyzer to help find the errors. It will be corrected using information from the Analyzer and re-run. The demonstration program is one which went through the final stages of debugging using the Analyzer. ICSE5 attendees will be able to observe the execution of the Analyzer on the sample program via a terminal connected by phone to the Institute for Computer Sciences and Technology VAX 11/780 system. Listings from the FORTRAN 77 Analyzer's run on the sample program will be available.

Demonstration time is about 20 minutes.

4. References

"FORTRAN 77 Analyzer", User's Manual, TRW Defense and Space Systems Group, One Space Park, Redondo Beach, CA 90278

"American National Standard Programming Language FORTRAN" ANSI X3.9-1978, American National Standard Institute, New York, 1978

Lyon and Stillman, "A FORTRAN Analyzer", NBS Technical Note 849, National Bureau of Standards, 1974

"FORTRAN 77 Analyzer", Maintenance Manual, TRW Defense and Space Systems Group, One Space Park, Redondo Beach, CA 90278

5. Place and Time

Station	Day	Time
5	Thursday, March 12	9 AM to 3 PM

6. Demonstrator

Patricia Powell
National Bureau of Standards
BLDG 225 RM A265
Washington, D.C. 20234
(301) 921-3485

Pat Powell received her undergraduate degree in Latin-American Studies from Smith College and her Master's degree in Computer Science from the University of Maryland. At NBS, she works in the area of Software Quality Assurance. Prior to NBS, she worked in AI and in the Research and Development of Programming Languages.

SUMMARY TOTALS

MODULE NAME	NUMBER OF STATEMENTS	PERCENT STATEMENT COVERAGE	NUMBER OF COMMENTS	NUMBER OF SYNTAX ERRORS	INVOCATION FREQUENCY	NUMBER OF SEGMENTS	PERCENT SEGMENT COVERAGE
MAIN	119	Ø.95	66	Ø	2	53	Ø.96
SHUFFL	17	1.ØØ	8	Ø	2	3	1.ØØ
DEAL	12	1.ØØ	5	Ø	26	3	1.ØØ
PLAY	25	1.ØØ	1Ø	Ø	94	7	Ø.86
EVALC	2Ø	Ø.ØØ	5	Ø	Ø	4	Ø.ØØ
MATCH	2Ø	1.ØØ	8	Ø	94	8	1.ØØ
LARGE	12	Ø.ØØ	5	Ø	Ø	4	Ø.ØØ
RMHAN	17	1.ØØ	5	Ø	96	5	1.ØØ
RMCFBD	17	1.ØØ	5	Ø	52	4	1.ØØ
BUILD	43	Ø.97	9	Ø	5Ø	14	Ø.93
DISCAR	26	1.ØØ	11	Ø	46	9	1.ØØ

SUMMARY TOTALS

MODULE NAME	NUMBER OF STATEMENTS	PERCENT STATEMENT COVERAGE	NUMBER OF COMMENTS	NUMBER OF SYNTAX ERRORS	INVOCATION FREQUENCY	NUMBER OF SEGMENTS	PERCENT SEGMENT COVERAGE
PROGRM	328	Ø.89	137	Ø	462	114	Ø.89

PROGRAM CALL TREE

```
      1       2       3       4       5       6       7       8       9       1Ø
 1    MAIN
 2            SHUFFL
 3                    RAN
 4            DEAL
 5            DISCAR
 6                    RMHAN
 7            RMCFBD
 8            PLAY
 9                    DISCAR (  5)
1Ø                    RMCFBD
11                    MATCH
12                    BUILD
13                            DISCAR (  5)
14                            RMHAN
15                    RMHAN
16   EVALC
17   LARGE
```

SEGMENT EXECUTION FREQUENCIES - CURRENT

	0	1	2	3	4	5	6	7	8	9
	MAIN									
0X		1	4	4	4	8	8	4	4	52
1X	52	4	4	4	16	16	4	4	3	12
2X	12	17	68	68	3	14	56	6	50	1
3X	49	3	10	3	7	2	2	0	2	5
4X	5	8	8	47	49	47	49	56	14	3
5X	4	0	4	1						
	SHUFFL									
5X					1	51	51			
	DEAL									
5X								13	52	52
	PLAY									
6X	47	0	47	188	25	22	47			
	EVALC									
6X								0	0	0
7X	0									
	MATCH									
7X		47	117	340	28	24	4	316	316	
	LARGE									
7X										0
8X	0	0	0							
	RMHAN									
8X				48	21	27	41	41		
	RMCFBD									
8X									26	6
9X	20	33								
	BUILD									
9X			25	100	52	136	337	3	3	0
10X	334	334	136	22	3	3				
	DISCAR									
10X							23	39	33	5
11X	28	30	3	30	23					

SEGMENTS NOT EXECUTED

37	51	61	67	68	69	70	79	80	81
82	99								

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . CODE INPUT
. . . FORTRAN
. . . FORTRAN 77
FUNCTION
. TRANSFORMATION
. . INSTRUMENTATION
. STATIC ANALYSIS
. . STATISTICAL ANALYSIS
. . . PROFILE GENERATION
. DYNAMIC ANALYSIS
. . COVERAGE ANALYSIS
. . TRACING
. . ASSERTION CHECKING
. . . RUN-TIME ASSERTION CHECKING
. . TUNING
OUTPUT
. USER OUTPUT
. . TABLES
. . LISTINGS
. MACHINE OUTPUT
. . SOURCE CODE OUTPUT
. . . FORTRAN 77

IMPLEMENTATION LANGUAGE: FORTRAN 77

TOOL PORTABLE: YES

TOOL AVAILABLE: YES, PUBLIC DOMAIN: YES

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): AVAILABLE FROM NTIS (FALL 81)

CONTACT: JOHN BARKLEY, NATIONAL BUREAU OF STANDARDS, TECH BLDG, A265, WASHINGTON, DC, 20234, USA, 301-921-3485

NBS SOFTWARE TOOLS DATABASE

Raymond C. Houghton, Jr. and Karen A. Oakley

1. Introduction

The NBS Software Tools Database is a compilation of data on the availability of over 250 software development and testing tools. The data that has been compiled has been placed into a relational database using Pascal/R, a language that extends Pascal by a data relation. The database allows for information retrieval on tool features, languages, developers, documentation, hardware and software requirements, availability, publications, and contacts. The information in a database of tools can be used by management to develop a software engineering methodology.

It has been concluded in recent reports that modern software tools can offer the software engineering community the following:

-- Better management control of computer software development, operation, maintenance, and conversion.

-- Lower costs for computer software development, operation, maintenance, and conversion.

-- Feasible means of inspecting both contractor-developed and in-house - developed computer software for such quality indications as conformance to standards and thoroughness of testing.

There are several reasons for compiling information on software tools. The first is to aid NBS efforts to develop guidelines and standards that will improve the quality of Federal software through the use of available tools. The second reason is to provide a means by which persons in the Federal Government and elsewhere can determine what tools are available and what their capabilities are. The third reason is to provide a means by which tool researchers can determine what tools are currently under development so that knowledge may be shared and duplication of effort can be avoided.

2. Summary

The following is a list of data elements that are stored in the database for each tool. Following each title is a brief summary of the information that is stored. Since the data base is relational, many of the elements can be repeated for a given tool.

ACRONYM: Acronym or other short name.

TITLE: Title of tool.

DOCUMENTATION: Types of available documentation.

DOC LENGTH: Extent of available documentation (page count).

REFERENCES: Articles or publications that discuss the tool and are readily available in the open literature.

DEVELOPER: Developer(s) of tool.

CONTACT: Contact(s) for more information about the tool.

INFORMATION SOURCE: Source(s) of the information contained in the database.

3. Scenario

Associated with the database is a tool that retrieves information and generates documentation for specific software tools or all software tools within the database. This tool is divided into five functional procedures each performing a specific task such as: (1) allowing a user to input the acronym of a tool, (2) directing docmention to a file, (3) retrieving information by tool features, (4) sorting the acronyms of tools, and (5) printing requested information to a terminal. A technical guide will be available to illustrate the application of the NBS Software Tools Database. Sample listings of input and output will also be available. Demonstration of the NBS Tool Database will be done interactively on a terminal connected by phone to the NBS DECSYSTEM-10 over the ARPANET.

4. References

Houghton and Oakley, 'NBS SOFTWARE TOOLS DATABASE', NBSIR 80-2159, November 80.

5. Station, Day and Time

Station 5, Thursday, March 12, 9AM - 3PM

6. Demonstrator

Raymond C. Houghton, Jr., National Bureau of Standards, Tech Bldg, Room A265, Washington, DC 20234, (301) 921-3485. Houghton is a computer scientist at the Bureau's Institute for Computer Sciences and Technology where he is in charge of the Tools Project. The purpose of this project is to aid the development of guidelines and standards that will improve the quality of Federal software through the use of available software development tools.

```
                    NBS SOFTWARE TOOLS DATABASE

WHAT PROCEDURE :        ENTER        PRINT          RETRIEVE    SORT
                        TYPE OR NOTHING?
RETRIEVE
ENTER FEATURE DESCRIPTION
>COVERAGE ANALYSIS
>

          30 TOOLS FOUND

TYPE TAXONOMY, YES OR NO? NO
WHAT PROCEDURE :        ENTER        PRINT          RETRIEVE    SORT
                        TYPE OR NOTHING?
RETRIEVE
ENTER FEATURE DESCRIPTION
>ASSERTION CHECKING
>

           4 TOOLS FOUND

TYPE TAXONOMY, YES OR NO? NO
WHAT PROCEDURE :        ENTER        PRINT          RETRIEVE    SORT
                        TYPE OR NOTHING?
RETRIEVE
ENTER FEATURE DESCRIPTION
>ASSERTION CHECKING ^ COVERAGE ANALYSIS
>

          32 TOOLS FOUND

TYPE TAXONOMY, YES OR NO? NO
WHAT PROCEDURE :        ENTER        PRINT          RETRIEVE    SORT
                        TYPE OR NOTHING?
RETRIEVE
ENTER FEATURE DESCRIPTION
>ASSERTION CHECKING & COVERAGE ANALYSIS
>

           2 TOOLS FOUND

TYPE TAXONOMY, YES OR NO? NO
WHAT PROCEDURE :        ENTER        PRINT          RETRIEVE    SORT
                        TYPE OR NOTHING?
NOTHING

EXIT
```

```
                    NBS SOFTWARE TOOLS DATABASE

WHAT PROCEDURE :          ENTER          PRINT              RETRIEVE     SORT
                          TYPE OR NOTHING?
ENTER
ENTER '$ALL' FOR LISTING OF ALL TOOLS
ACRONYM OR SHORT TITLE ? FTN-77 ANALYZER
ACRONYM OR SHORT TITLE ? TOOLS DATABASE
ACRONYM OR SHORT TITLE ?
WHAT PROCEDURE :          ENTER          PRINT              RETRIEVE     SORT
                          TYPE OR NOTHING?
TYPE
WHAT SHOULD BE TYPED :    ABSTRACT       BIBLIOGRAPHY       CONTACT      DOCUMENTATION
                          ENVIRONMENT    HARDWARE           IDENTITY     LANGUAGE
                          MAKER          PROGRESS           REFERENCES   SOFTWARE
                          TAXONOMY       USEABILITY OR NOTHING?
IDENTITY
ACRONYM           TITLE
                          CLASSIFICATION
-------------------------------------------------------------------------------
FTN-77 ANALYZER   NBS FORTRAN-77 ANALYZER
                          SOURCE PROGRAM ANALYSIS AND TESTING

TOOLS DATABASE    NBS SOFTWARE DEVELOPMENT TOOLS DATABASE
                          SOFTWARE MANAGEMENT CONTROL, AND MAINTENANCE

WHAT SHOULD BE TYPED :    ABSTRACT       BIBLIOGRAPHY       CONTACT      DOCUMENTATION
                          ENVIRONMENT    HARDWARE           IDENTITY     LANGUAGE
                          MAKER          PROGRESS           REFERENCES   SOFTWARE
                          TAXONOMY       USEABILITY OR NOTHING?
ENVIRONMENT
ACRONYM                TOOL PORTABLE      TOOL SIZE
-------------------------------------------------------------------------------
FTN-77 ANALYZER           YES

TOOLS DATABASE             NO

WHAT SHOULD BE TYPED :    ABSTRACT       BIBLIOGRAPHY       CONTACT      DOCUMENTATION
                          ENVIRONMENT    HARDWARE           IDENTITY     LANGUAGE
                          MAKER          PROGRESS           REFERENCES   SOFTWARE
                          TAXONOMY       USEABILITY OR NOTHING?
PROGRESS
ACRONYM             STAGE OF DEVELOPMENT    DATE OF DEVELOPMENT (YYMMDD)
-------------------------------------------------------------------------------
FTN-77 ANALYZER     IMPLEMENTED                    810500

TOOLS DATABASE      IMPLEMENTED                    801000
```

```
WHAT SHOULD BE TYPED :  ABSTRACT      BIBLIOGRAPHY      CONTACT      DOCUMENTATION
                        ENVIRONMENT   HARDWARE          IDENTITY     LANGUAGE
                        MAKER         PROGRESS          REFERENCES   SOFTWARE
                        TAXONOMY      USEABILITY OR NOTHING?
MAKER
ACRONYM
     DEVELOPER
-------------------------------------------------------------------------------
FTN-77 ANALYZER
     TRW, INC.

TOOLS DATABASE
     NATIONAL BUREAU OF STANDARDS

WHAT SHOULD BE TYPED :  ABSTRACT      BIBLIOGRAPHY      CONTACT      DOCUMENTATION
                        ENVIRONMENT   HARDWARE          IDENTITY     LANGUAGE
                        MAKER         PROGRESS          REFERENCES   SOFTWARE
                        TAXONOMY      USEABILITY OR NOTHING?
LANGUAGE
ACRONYM          LANGUAGE              DIALECT
-------------------------------------------------------------------------------
FTN-77 ANALYZER  FORTRAN               FORTRAN 77

TOOLS DATABASE   PASCAL                R

WHAT SHOULD BE TYPED :  ABSTRACT      BIBLIOGRAPHY      CONTACT      DOCUMENTATION
                        ENVIRONMENT   HARDWARE          IDENTITY     LANGUAGE
                        MAKER         PROGRESS          REFERENCES   SOFTWARE
                        TAXONOMY      USEABILITY OR NOTHING?
NOTHING
WHAT PROCEDURE :        ENTER         PRINT             RETRIEVE     SORT
                        TYPE OR NOTHING?
NOTHING

EXIT
```

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . DATA INPUT
FUNCTION
. STATIC ANALYSIS
. . MANAGEMENT
. . . MANAGEMENT PLANNING
OUTPUT
. USER OUTPUT
. . TABLES

IMPLEMENTATION LANGUAGE: PASCAL R

TOOL PORTABLE: NO

COMPUTER (OTHER HARDWARE): DECSYSTEM-10

OS (OTHER SOFTWARE): TOPS-10

TOOL AVAILABLE: YES, PUBLIC DOMAIN: YES

CONTACT: RAYMOND C. HOUGHTON, NATIONAL BUREAU OF STANDARDS, TECH BLDG, A265, WASHINGTON, DC, 20234, USA, 301-921-3485

SRIMP

SOFTWARE REQUIREMENTS INTEGRATED MODELING PROGRAM

A Demonstration of a Front End Modeling Tool

Stephanie White

1. INTRODUCTION

This proposal is in response to a request for demonstrations of software engineering tools at the 5th International Conference on Software Engineering. Grumman Aerospace Corporation will be demonstrating the capabilities of SRIMP and its function in a software engineering environment. SRIMP, a front end modeling tool, is part of Grumman's Software Life Cycle Development System (SOLID). SOLID is a system of methodologies and automated life cycle tools that supports an environment in which software can be designed, developed, implemented, tested and maintained.

2. Description of SRIMP

The Software Requirements Integrated Modeling Program (SRIMP) aids in the conceptual and requirements definition phases of the software development process.

The SRIMP methodology was developed by synthesizing the better features of a number of widely accepted requirements techniques, augmented with a number of Grumman developed innovations. The resultant system provides the user with a formal language to express specifications through the identification of objects and acceptable relationships. Top-down decomposition is enforced by a method which results in structured intra system interfaces, thus minimizing model complexity.

Human factors engineering, which played a major role in the development of SRIMP, was responsible for the production of an easy-to-use tool. The user is led through a series of prompts where constant monitoring is performed to assure that the user provides valid responses. Easy-to-understand diagnostics are provided when incomplete or inconsistent data is entered.

The requirements are stored in a SRIMP data base which lends itself to efficient maintenance. Outputs of SRIMP are structure reports, function reports, hierarchy charts and functional flow diagrams. The visual representation of a model gives the user an overall hierarchical view of the system at a glance. Data flow, as well as function flow, is visible in diagrams of a function and its immediate offspring. Diagrams are automatically produced from the data base and can be displayed on a CRT or produced on plotters, slides or microfilm.

The SRIMP data base can be automatically translated into an input source file for PSL/PSA. Thus the benefits of PSL/PSA reports, which are an excellent aid in documentation, are available. Under PSL/PSA the model can be further defined, documented and analyzed. In addition, a detranslator has been developed which accesses the requirements data base in order to enhance PSL/PSA with Grumman graphics.

3. The SRIMP Demonstration

The demonstration of SRIMP will consist of a viewgraph presentation followed by an interactive session. The narrative overview of SRIMP will address the following areas:

1. the usefulness of a tool such as SRIMP during front end modeling

2. the SRIMP methodology

3. the system features

During the interactive session, an existing SRIMP data base will be modified, checked for consistency and completeness, and translated into PSL. Hierarchy charts and functional flow diagrams will be generated from the data base. The demonstration will take 30 minutes and will be repeated at 9 AM, 10 AM, 11 AM, 2 PM and 3 PM.

4. SRIMP Literature

H. Barina et al, "Automated Software Design," Proceedings, IEEE Computer Society's Third International Conference and Applications Conference, November 1979.

L. Fabiano and J. McCarthy, "Software Life Cycle Development (SOLID)", will appear in Proceedings, NAECON '81, May 1981.

A copy of the presentation viewgraphs and scenario will be distributed upon request.

5. Demonstration Station, Day and Time

Station 7, Thursday, March 12 from 9 AM until 12 Noon and from 2 PM until 5 PM.

6. The DEMONSTRATOR

Stephanie White

Education

BA, Mathematics, Hunter College
MS, Mathematics, New York University
MS, Computer Science, Polytechnic Institute of New York
PhD candidate, Computer Science, Polytechnic Institute of New York

6. The DEMONSTRATOR (Cont'd)

Stephanie White

Experience

Ms. White is assigned to the Grumman Software Systems Department's Technology Group and is principal investigator of software requirements specification methods and tools. She is responsible for developing methodologies related to problem analyzers and integrated systems data bases. Among her recent achievements have been establishment of computerized graphics modeling tools that produce hierarchical structure and IDEF charts. She has also designed a translator to change a Grumman model to an equivalent Problem Statement Language/Problem Statement Analyzer (PSL/PSA) model in PSL source code. The model was created using an interactive modeling tool developed at Grumman.

Ms. White has developed a program to access the PSA data base in order to enhance PSL/PSA with Grumman graphics.

She was also employed by C.W. Post College where she developed course content, served on the curriculum committee and taught graduate and undergraduate courses in mathematics and computer science.

Society Membership

ACM, IEEE, Phi Beta Kappa, Pi Mu Epsilon, Woodrow Wilson Fellow, Women and Mathematics.

Papers/Publications

"Automated Structured Design", Proceedings, COMPSAC conference, IEEE, 1979
"Student Study Guide for Calculus with Analytic Geometry", Macmillan Publishing Co., 1978.

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . VHLL INPUT
. . . MODEL DESCRIPTION
FUNCTION
. TRANSFORMATION
. . TRANSLATION
. . EDITING
. . FORMATTING
. STATIC ANALYSIS
. . DATA FLOW ANALYSIS
. . CONSISTENCY CHECKING
OUTPUT
. USER OUTPUT
. . GRAPHICS
. . . HIERARCHICAL TREE
. . . ACTIVITY DIAGRAM
. . LISTINGS
. MACHINE OUTPUT
. . VHLL OUTPUT
. . . PSL

IMPLEMENTATION LANGUAGE: FORTRAN, PL/1

TOOL PORTABLE: YES

TOOL AVAILABLE: YES, PUBLIC DOMAIN: NO

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): CONTACT GRUMMAN AEROSPACE CORP. FOR DETAILS

TOOL SUPPORTED: YES, TOOL SUPPORT: GRUMMAN AEROSPACE CORP.

CONTACT: STEPHANIE WHITE, GRUMMAN AEROSPACE CORP., TECHNOLOGY DIVISION, M/S A02/35, BETHPAGE, NY, 11714, USA, 516-575-6493

AUTOMATED DESIGN BY OBJECTIVES ("AUTO-DBO")

Summary of Tool.
AUTO-DBO is a research project with the objective of exploring the limits of our understanding of the software engineering process.

The principal method for exploration is to see how far we can instruct a computer to carry out the software design process.

We are also interested in seeing to what degree a computer can give other types of help in addition to "design" itself: for example as a structured text editor, helping evaluate consequences of changes, making multiple point changes with little effort, teaching the design process.

It should be stressed that we are speaking of high level "architectural" design. This includes goals or requirements (such as application functions and particularly attributes (resources and qualities). It includes technique specification, evaluation of alternative techniques , evaluation of alternative major sub-architectures (such as "which data base software ?"), design completeness evaluation, design incompleteness evaluation, design documentation inspection statistics recording and reporting, and incremental delivery planning.

It does not include low level architecture concepts such as planning of the detailed logic of single modules : in other words execution sequence is of no concern in detail, although we might well consider various global rules which influence the programmed sequence or structure, such as "use decision table logic" (as a technique specification).

It should be stressed that the DBO method, and the AUTO-DBO tool is capable of considering the total systems environment (organizational forms, documentation design, data structures and bases, hardware, and the integration of these with software.

The present state of the tool is the first demonstratable stage of development. About 30% of the plans have been implemented. The tool is available for use and experimentation. It can be used for real projects, but it is not presently capable of making a design decision (translating

for example a reliability requirement into a set of system architecture techniques such as " use distinct software on all updating logic").

In very simple terms the present system is a highly structured and integrated text editor. It is the framework for design automation (the next stage). We are inviting interested parties to participate in future design stages in various roles (user, developer etc.). The present system is the only one we know of which explicitly specifies the required system attributes in hierarchical and measurable for, and then integrates the evaluation of the achievement of these attributes into the later design evaluation stages.

The human interface has been designed to allow easy learning, ease of operation, and highly reliable use. This is already well implemented. Among the demonstrable features at present is the use of the keyboard as a joystick for both moving a window over design documentation forms, and for moving up and down hierarchies of documentation. A keyword search capability allows complete or selective searches of design documentation, and immediate jumping to the keyword's portion of the design documentation.

The system is operational using UCSD Pascal on Apple II with two disk drives. One drive is used for logicware, one for design documentation. Extensive systems description can be accomplished by using a hierarchy of diskettes, so that we are not limited by the small space available on Apple 5" diskettes. The system will be shortly implemented on CYBER Pascal, and we have had requests from parties interested in implementing it on IBM and Nippon Electric computers. The advantage of the Apple implementation is that it can be used independently to support the design tool for a project group for example.

SCENARIOS

We expect to demonstrate most of the features using various project files, including the one which describes the design of AUTO-DBO itself.
Visitors will be welcome to try hands on if they like.
We hope to show enough so that visitors will see the potential of such a design tool, and so that some visitors will decide to either pursue similar developments or to join our informal project.

REFERENCES:

1. GILB: Computerware Technoscopes. 300 page manuscript description of the Design by Objectives methods as applied to complex system perception and evaluation. To be published by North-Holland. Available until then for copying from author.

2. AUTO-DBO DOCUMENTATION (Oct 9 1980 Version) 60 page documentation of the AUTO-DBO system initial implementation and global design (including future enhancements). By Lech Krzanik, Univ. of Krakow.

3. Pascal Program Listing.

4. Diskettes with source and executable programs , as well as demo project diskettes are available for copying upon request for contributors to the project.

5. A number of papers and articles on DBO are available in addition on request.

DEMONSTRATOR : Tom Gilb, Iver Holtersvei 2, N-1410 Kolbotn, Norway will demonstrate the system. It is unlikely, but not impossible that the author of the software and most of the detailed design, Lech Krzanik (WŁOCZKOW 18/10, PL-30103 KRAKOW, POLAND. Tel (48 94) 25817) will also be able to attend. But he will be available for consultation by mail and telephone. It is also expected that he will provide additional documentation for the Tool Fair demo.

TOM GILB: is a private and independent consultant, teacher and author living in Norway and working on all continents. He is the author of Software Metrics, and (with Gerald M. Weinberg) Humanized Input. His major professional activities center around development of the Design by Objectives method, with particular reference to both logicware and dataware engineering.
LECH KRZANIK works at University of Krakow at the new Computer Science Institute. He is completing his Dr. Thesis, partly based on the AUTO-DBO project, and will continue work in that area.

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . VHLL INPUT
. . . DESIGN SPECIFICATION
FUNCTION
. TRANSFORMATION
. . EDITING
. . TRANSLATION
. STATIC ANALYSIS
. . CONSISTENCY CHECKING
. . COMPLETENESS CHECKING
. . INTERFACE ANALYSIS
OUTPUT
. USER OUTPUT
. . USER-ORIENTED TEXT
. . . DOCUMENTATION
. MACHINE OUTPUT
. . VHLL OUTPUT
. . . DESIGN SPECIFICATION

IMPLEMENTATION LANGUAGE: PASCAL UCSD PASCAL

TOOL PORTABLE: NO

COMPUTER (OTHER HARDWARE): APPLE II (TWO FLOPPY DISK DRIVES)

CONTACT: TOM GILB, INDEPENDENT EDP CONSULTANT, IVER HOLTERSVEI 2, KOLBOTN, N-1410, NORWAY, 47 2-80 16 97

THE UCSD p-SYSTEM VERSION IV.0
A PORTABLE SOFTWARE DEVELOPMENT SYSTEM

A Software Engineering Tool Demonstration

Mark Overgaard and Joan Giannetta

I. INTRODUCTION

We will be demonstrating Version IV.0 of the UCSD p-System, developed and distributed by Softech Microsystems, San Diego, California.

II. SUMMARY OF THE UCSD p-SYSTEM

The UCSD p-System is a stand-alone program development and execution environment for small computers. Its facilities include text editors and file management utilities, as well as compilers (UCSD Pascal , FORTRAN-77 and BASIC), macro cross-assemblers and a linkage editor.

The p-System provides a portable software environment independent from the host processor and its peripheral devices. The foundation for this portability is the UCSD P-machine. It is a simple idealized stack computer which can be implemented by direct hardware support or by an interpreter executing in the machine language of the host computer. All system software is written in UCSD Pascal, compiled to P-code and then executed by the P-machine.

III. THE DEMONSTRATION

The demonstration of the UCSD p-System includes an introduction to the p-System facilities and a discussion on how program portability has been attained.

High level language and assembly language routines will be edited, translated and linked. Transparent execution of the resulting code files on machines with dissimilar physical processors will be demonstrated.

The demonstration will take approximately one half hour and will be followed by a question and answer period. Several microcomputers will be available for the attendees to gain hands-on experience.

IV. UCSD p-SYSTEM LITERATURE AND RELATED DOCUMENTS

"UCSD Pascal Users Manual Version IV.0" - A basic reference guide for use of the UCSD p-System and UCSD Pascal.

"UCSD p-System Architecture Guide" - An extensive description of the UCSD P-machine and p-System architecture.

"UCSD p-System Installation Guide" - A guide to bringing up, "bootstrapping", the

UCSD p-System.

"FORTRAN User Reference Manual" - A guide for using the UCSD p-System ANSI-77 subset FORTRAN compiler.

"BASIC User Reference Manual" - A guide for using the UCSD p-System BASIC compiler.

"Beginner's Guide for the UCSD Pascal System", Dr. Kenneth L. Bowles - A beginner's look at the UCSD Pascal System.

"Problem Solving Using Pascal", Dr. Kenneth L. Bowles - A non-numerical approach to problem solving using computers.

In addition, the following papers will be of interest:

Overgaard, Mark, "UCSD Pascal : A Portable Software Environment for Small Computers," AFIPS - Conference Proceedings, Volume 49, AFIPS Press, Arlington, Va. 22209.

Irvine, C.A., "UCSD System Makes Programs Portable," Electronic Design, 16 August 1980.

Bowles, K.L., "A (Nearly) Machine Independent Software System for Micro and Mini Computers," Byte, May 1978.

V. LOCATION AND SCHEDULE OF DEMONSTRATION

Station 8, Thursday, 12 March 1981, from 9:00 a.m. to 4:00 p.m.

VI. THE DEMONSTRATORS

MARK OVERGAARD

Mark Overgaard is the Manager of Software Development at Softech Microsystems. He has played a principal technical role throughout the development and evolution of the UCSD p-System, working first as a graduate student with Dr. Kenneth L. Bowles at the University of California, San Diego, and more recently at SofTech Microsystems.

JOAN GIANNETTA

Joan Giannetta is a Marketing Account Manager for SofTech Microsystems. She previously worked as a systems programmer, maintaining and enchancing a P-code interpreter and p-System facilities for Dr. Kenneth L. Bowles at the Universtiy of California, San Diego.

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . CODE INPUT
. . . FORTRAN
. . . FORTRAN 77
. . . PASCAL
FUNCTION
. TRANSFORMATION
. . EDITING
. . TRANSLATION
OUTPUT
. USER OUTPUT
. . LISTINGS

IMPLEMENTATION LANGUAGE: PASCAL UCSD

TOOL PORTABLE: YES

TOOL AVAILABLE: YES, PUBLIC DOMAIN: YES

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): LICENSE

TOOL SUPPORTED: YES, TOOL SUPPORT: SOFTECH MICROSYSTEMS

CONTACT: SOFTECH MICROSYSTEMS, 9494 BLACK MOUNTAIN ROAD, SAN DIEGO, CA, 92126, USA, 714-578-6105

The Microprocessor Software Engineering Facility

SofTech, Inc.

1. Introduction

This proposal is in response to the call for software engineering tool developers to demonstrate their accomplishments at the 5th International Conference on Software Engineering in San Diego, March 9-12, 1981.

SofTech will demonstrate the Microprocessor Software Engineering Facility (MSEF), a tool package initially developed under contract to ITT Defense Communications Division and subsequently extended by SofTech.

2. Summary of the MSEF

The Microprocessor Software Engineering Facility (MSEF) is an integrated set of software tools to support the development and maintenance of microcomputer software. The MSEF is hosted on a PDP-11 computer under the UNIX operating system but can support the production of software for many different microcomputers. The MSEF Change Control Library promotes defining, updating, and integrating parts of a software configuration, isolation of user work environments, and version control. The MSEF supports the organized testing of software components by associating test scenarios and test results with the components to be tested. The MSEF also provides automatic change logging with a configuration audit trail. A macro assembler supporting structured language constructs and a compiler for "C" are included in the tool complement.

3. Scenario of an MSEF Demonstration

The MSEF demonstration consists of the following:

a) A 10 minute introduction to the MSEF and the demonstration.

b) The master configuration is examined, the log and other features are described.

c) Using the MERGE command, a local configuration is created to allow changes to be developed.

d) The configuration is linked and executed. The program consists of 5 modules in "C" and user written tools in the MSEF command language. Both the "C" program and the tools are executed. The "C" program generates a horizontal line of large letters corresponding to the string supplied as an input argument.

e) The program is modified to print the letters vertically. A record of the changes is accumulated automatically by the MSEF. The various audit trails and logs are examined to demonstrate the operation of the configuration management mechanisms.

f) The COMPARE command is used to compare the altered local configuration with the master configuration. All changes applied to the

f) local configuration are readily visible.

g) The MERGE command is used to install the altered version back in the master configuration.

4. MSEF References

Eanes, Hitchon, Thall, and Brackett; "An Environment for Producing Well-Ingineered Microcomputer Software"; Proceedings of 4th ICSE, September 1979, pp 386-398.

5. Station, Day, and Time

Station 10, Thursday, March 12, from 9:00 a.m. until 5:00 p.m.

6. The Demonstrators

Rich Thall

Rich Thall is a Systems Consultant with SofTech, Inc. He was principal designer and implementor of the MSEF change control tools. He is presently leading the Ada environment development team in SofTech's Ada program. He holds B.S. and M.S. degrees from the University of Wisconsin, where he also served as a Research Assistant for the Information Systems Design and Optimization System Project (ISDOS).

Gail Anderson

Gail Anderson is a Software Engineer with SofTech MicroSystems. She is currently working on the UCSD p-System operating system. She holds a B.A. from UCSD and a M.A. from San Diego State University - both in linguistics. She was on the Development team on MSEF enhancements done at SofTech MicroSystems. Before joining SofTech, she was a UNIX Systems Programmer at the University of California in Santa Barbara.

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . CODE INPUT
FUNCTION
. TRANSFORMATION
. . EDITING
. . TRANSLATION
. STATIC ANALYSIS
. . MANAGEMENT
. . . VERSION CONTROL
. . . CHANGE CONTROL
. . . TEST DATA MANAGEMENT
. . COMPARISON
OUTPUT
. USER OUTPUT
. . LISTINGS
. MACHINE OUTPUT
. . INTERMEDIATE CODE

IMPLEMENTATION LANGUAGE: C

COMPUTER (OTHER HARDWARE): PDP-11

OS (OTHER SOFTWARE): UNIX

TOOL AVAILABLE: YES, PUBLIC DOMAIN: NO

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): LEASE

CONTACT: VIC VOYDOCK, SOFTECH, INC., 460 TOTTEN POND ROAD, WALTHAM, MA, 02154, USA, 617-890-6900

IFTRAN™

A PREPROCESSOR FOR FORTRAN

A SOFTWARE ENGINEERING TOOL DEMONSTRATION

Sabina H. Saib, Jeoffrey P. Benson,
Carolyn Gannon, and William R. DeHaan

1. Introduction

This proposal is in response to the call for software engineering tool developers to demonstrate their accomplishments at the 5th International Conference on Software Engineering in San Diego, March 9-12, 1981.

We will be demonstrating IFTRAN™, developed in The Software Workshop™ by General Research Corporation, Santa Barbara, California.

2. Summary of IFTRAN™

IFTRAN™ is an extension of FORTRAN that simplifies structured programming and top-down design, aids systematic testing, supports a practical application of formal verification techniques, and aids in software fault tolerance.

IFTRAN™ has a convenient syntax for writing structured programming control constructs. The IFTRAN™ preprocessor translates the IFTRAN™ statements into standard FORTRAN while passing all other statements unchanged to an output file which may then be compiled by a FORTRAN compiler. A listing which automatically indents the source program is also generated.

IFTRAN™ can be used as a program design language when English is used in the constructs instead of FORTRAN. The indented listing is valuable for design reviews.

As an aid to testing, the preprocessor can be directed to automatically insert calls to data collection routines into the program so that program execution sequence can be traced and measured. A standard set of analysis routines included with the preprocessor provide execution analysis reports.

Assertions from formal program verification may be incorporated in an IFTRAN™ program to provide documentation or to provide automatic checks on legitimate ranges of variables. Programmers may specify corrective actions (FAIL blocks) to be taken when assertions are violated.

3. A Scenario of an IFTRAN™ Demonstration

A demonstration of IFTRAN™ consists of three parts. The first part is a demonstration of the PROGRAM DESIGN LANGUAGE feature (see SAMPLE OUTPUT, Figure 1). The second part is a demonstration of the BLOCK COPYING, MACRO, INSTRUMENTATION (identification of DD-Paths), TESTBOUND, STATEMENT COMMENTING, EXECUTABLE ASSERTIONS, and INDENTED LISTING features (see SAMPLE OUTPUT, Figure 2). The third part is a demonstration of the types of reports provided by the ANALYZER PROGRAM as a result of the INSTRUMENTATION probe processing (See SAMPLE OUTPUT, Figure 3).

The demonstration will take one hour. We will repeat it as often as necessary. We can handle about 10 people in each interactive demonstration.

4. IFTRAN™ Literature

User's Manual

A description of the uses of the IFTRAN™ syntax and the IFTRAN™ preprocessor.

Collected Papers

A collection of papers written by members of the Software Quality Department.

5. Station, Day, and Time

Forum, Tuesday, March 10, from 11:00 am until 12:30 pm.
Forum, Wednesday, March 11, from 9:00 am until 10:30 am
Station 9, Thursday, March 12, from 9:00 am to 5:00 pm.

6. The Demonstrators

Sabina H. Saib is Director of the Software Quality Department at General Research. She received a BS and a PhD in Engineering from UCLA and an MSEE from the University of Maryland. She has published several papers on the verification and validation of software. Her current work is in the automated verification of embedded systems.

Carolyn Gannon a member of the Software Quality Department at General Research. She received her BS and MS with a computer science specialty from the University of California, Santa Barbara. She is active in the JOVIAL users group and is leading the development of an automated verification system for JOVIAL J73. She has published several papers on test techniques and automated verification systems.

Jeoffrey P. Benson received his BA degree from California State University, Fresno, and his MSEE and PhD degrees from the University of California, Santa Barbara. Since 1973 Dr. Benson has been at General Research Corporation, where he has done research in software verification. He has also taught courses in compiler design and construction at the University of California, Santa Barbara. His current research interests include compiler-compilers and biomedical computing. Dr. Benson is a member of IEEE and ACM.

William R. DeHaan is Software Marketing Director at General Research. He received his BS from Fairleigh Dickinson University and is currently active in the creation, preparation, and marketing of proprietary software. Mr. DeHaan is a member of ACM.

```
V I F T R A N-- A PROPRIETARY SOFTWARE TOOL FROM THE SOFTWARE WORKSHOP AT GENERAL RESEARCH CORPORATION --V I F T R A N
  SEQ NEST SOURCE                PROGRAM KEYTST                                                          PAGE     1

    1           PROGRAM KEYTST
    2      C.KEYWORD=ON
    3           LOOP FOREVER
    4  1        .   FOR FIRST/NEXT CARD
    5           EXIT IF NO MORE CARDS
    6  2        .   .   FOR CARD COLUMNS 1 THROUGH 10
    7  3        .   .   .   IF CURRENT CARD COLUMN CONTAINS A SPECIAL CHARACTER
    8  4        .   .   .   .   CALL ROUTINE TO ENCRIPT THE CHARACTER
    9  3        .   .   .   END IF
   10  2        .   .   END FOR
   11  2        .   .   SAVE ENCRIPTED CARD FOR LATER OUTPUT
   12  1        .   END FOR
   13           END LOOP
   14           IF ANY CARDS WERE ENCRIPTED
   15  1        .   OUTPUT THE CARDS TO THE SAVE FILE FOR
   16  1        .   LATER PROCESSING
   17           END IF
   18           STOP
   19           END

IFTRAN STATISTICS
    19 CARDS READ
     0 ERROR(S) FOUND
```

FIGURE 1

IFTRAN™

PROGRAM DESIGN LANGUAGE Example

```
V I F T R A N-- A PROPRIETARY SOFTWARE TOOL FROM THE SOFTWARE WORKSHOP AT GENERAL RESEARCH CORPORATION --V I F T R A N
  SEQ NEST SOURCE                                                                                        PAGE     1

C.SAVE NAME
C     THIS IS MODULE ******.
C.END
C.SAVE TYPE
      COMMON /TYPE/ KIND
C.END
    1           PROGRAM IFTEST
    2      C.LIST=OFF
    4      C.LIST=ON
    5      C.MACRO NAME,/******/IFTEST/
    6      C     THIS IS MODULE IFTEST.
    7      C.END
    8      C.MACRO TYPE
    9           COMMON /TYPE/ KIND
   10      C.END
   11      C.INSTRUMENT=ON
   12      C.ENTRY=IFTEST
                                                          DD PATH     1 IS ENTER DECK
   13           KIND=0
   14           PRINT *,'BEGIN EXECUTION.'
   15           WHILE (KIND.NE.3)
                                                          DD PATH     2 IS WHILE DO,    3 IS WHILE EXIT
   16  1        .   PRINT 100
   17  1    100 .   FORMAT (' ENTER THE NUMBER TO BE SQUARE ROOTED.',/)
   18  1        .   READ *,SQRD
   19  1        .   IF (SQRD.LE.500)
                                                          DD PATH     4 IS IF TRUE,    5 IS IF FALSE
   20  2        .   .   KIND=1
   21  1        .   OR IF (SQRD.LE.1000)
                                                          DD PATH     6 IS OR IF TRUE,    7 IS OR IF FALSE
   22  2        .   .   KIND=2
   23  1        .   ELSE
   24  2        .   .   KIND=3   $WHEN SQRD .GT. 1000, STOP AFTER SQROOT
   25  1        .   END IF
   26  1        .   SQARD=SQROOT(SQRD)
   27  1        .   PRINT *,'  THE SQUARE ROOT OF ',SQRD,' IS ',SQARD
   28  1    C.TESTBOUND
   29           END WHILE
   30           PRINT *,'END EXECUTION'
   31           STOP
                                                          DD PATH     8 IS STOP
   32           END
```

FIGURE 2(a)

IFTRAN™

Example Usage of the

BLOCK COPYING, MACRO, INSTRUMENTATION,

TESTBOUND, STATEMENT COMMENTING, and INDENTED LISTING

Features

(MAIN PROGRAM)

```
V I F T R A N-- A PROPRIETARY SOFTWARE TOOL FROM THE SOFTWARE WORKSHOP AT GENERAL RESEARCH CORPORATION --V I F T R A N
  SEQ NEST SOURCE               FUNCTION SQROOT (ARG)                                                    PAGE     1

    1           FUNCTION SQROOT (ARG)
    2      C.LIST=OFF
    4      C.LIST=ON
    5      C.MACRO NAME,/******/SQROOT/
    6      C     THIS IS MODULE SQROOT.
    7      C.END
    8      C.MACRO TYPE
    9           COMMON /TYPE/ KIND
   10      C.END
   11      C.ASSERT=ON
   12      C.DEBUG=ON
   13      C.ENTRY=SQROOT
                                                          DD PATH     1 IS ENTER DECK
   14           BARG=ARG
   15           INITIAL (ARG.GT.0.0), FAIL (ERROR)
   16           SQROOT=SQRT(BARG)
   17           RETURN
                                                          DD PATH     2 IS RETURN
   18           BLOCK (ERROR)
                                                          DD PATH     3 IS ENTER BLOCK
   19  1        .   DEBUG (/REAL/ARG)
   20  1        .   BARG=0.0
   21  1        .   PRINT $,' FUNCTION SQROOT VALUE RETURNED EQUAL TO ZERO.'
   22           END BLOCK
                                                          DD PATH     4 IS EXIT BLOCK
   23           END
*****************************************************************************************************************************************

BLOCK CROSS-REFERENCE

BLOCK NAME                          DEFINED   INVOKED
----------                          -------   -------
ERROR                                 18        15
```

FIGURE 2(b)

IFTRAN™

Example Usage of the

BLOCK COPYING, MACRO, INSTRUMENTATION,

EXECUTABLE ASSERTIONS, and INDENTED LISTINGS

Features

(SUBPROGRAM)

```
..........compiling.........
..........linking...........
BEGIN EXECUTION.
ENTER THE NUMBER TO BE SQUARE ROOTED.

25
  THE SQUARE ROOT OF    25.00000      IS    5.000000
ENTER THE NUMBER TO BE SQUARE ROOTED.

-36

FOR MODULE SQROOT INITIAL FALSE AT STATEMENT   15

DEBUG FOR MODULE SQROOT AT STATEMENT   19
          ARG                =-36.00
 FUNCTION SQROOT VALUE RETURNED EQUAL TO ZERO.
  THE SQUARE ROOT OF   -36.00000      IS   0.0000000E+00
ENTER THE NUMBER TO BE SQUARE ROOTED.
1200
  THE SQUARE ROOT OF    1200.000      IS    34.64102
END EXECUTION
```

FIGURE 2(c)

IFTRAN™

Sample Output from the

Execution of the Programs in Figures 2(a) & 2(b)

Illustrating the EXECUTABLE ASSERTIONS Feature

RECORD OF DECISION TO DECISION (DD PATH) EXECUTION

MODULE $IFTEST $ TEST CASE NO. 1

DD PATH NUMBER	NO.	NOT EXECUTED	NUMBER OF EXECUTIONS -- NORMALIZED TO MAXIMUM .-------20.-------40.-------60.-------80.------100.			NUMBER OF EXECUTIONS
1			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	1	1	1
2			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	1	2	1
3	3	00000				
4			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	1	4	1
5	5	00000				
...	...	00000	... ...			
8	8	00000				

TOTAL NUMBER OF DD PATH EXECUTIONS = 3

TOTAL OF 5 NOT EXECUTED EXECUTED 3/ 8 PERCENT EXECUTED = 37.50

FIGURE 3(a)

IFTRAN™

Sample DETAILED INSTRUMENTATION REPORT

For Program In Figure 2(a)

RECORD OF DECISION TO DECISION (DD PATH) EXECUTION

MODULE $SOROOT $ TEST CASE NO. 1

DD PATH NUMBER	NO.	NOT EXECUTED	NUMBER OF EXECUTIONS -- NORMALIZED TO MAXIMUM .-------20.-------40.-------60.-------80.------100.			NUMBER OF EXECUTIONS
1			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	1	1	1
2			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	1	2	1
3	3	00000				
4	4	00000				

TOTAL NUMBER OF DD PATH EXECUTIONS = 2

TOTAL OF 2 NOT EXECUTED EXECUTED 2/ 4 PERCENT EXECUTED = 50.00

FIGURE 3(b)

IFTRAN™

Sample DETAILED INSTRUMENTATION REPORT

For Program In Figure 2(b)

TEST CASE	MODULE NAME	NUMBER OF D-D PATHS	SUMMARY -- THIS TEST: NUMBER OF INVOCATIONS	SUMMARY -- THIS TEST: D-D PATHS TRAVERSED	SUMMARY -- THIS TEST: PER CENT COVERAGE	CUMULATIVE SUMMARY: NUMBER OF TESTS	CUMULATIVE SUMMARY: INVOCATIONS	CUMULATIVE SUMMARY: TRAVERSED	CUMULATIVE SUMMARY: COVERAGE
1	IFTEST	8	1	3	37.50	1	1	3	37.50
	SOROOT	4	1	2	50.00	1	1	2	50.00
	$$ALL$$	12		5	41.67	1		5	41.67
2	IFTEST	8	0	2	25.00	2	1	3	37.50
	SOROOT	4	1	4	100.00	2	2	4	100.00
	$$ALL$$	12		6	50.00	2		7	58.33

FIGURE 3(c)

IFTRAN™

Sample SUMMARY INSTRUMENTATION REPORT

For Programs In Figures 2(a) & 2(b)

(Multiple Test Cases due to C. TESTBOUND In Figure 2(a))

MODULE NAME	TEST NUMBER	PATHS NOT HIT	LIST OF DECISION TO DECISION PATHS NOT EXECUTED
<IFTEST >	2	6	1 3 5 6 7 8
	CUMUL	5	1 3 5 6 7 8
<SOROOT >	2	0	
	CUMUL	0	

FIGURE 3(d)

IFTRAN™

Sample "NOT HIT" INSTRUMENTATION REPORT

For Programs In Figures 2(a) & 2(b)

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . CODE INPUT
. . . FORTRAN
. . . IFTRAN
. . VHLL INPUT
. . . IFTRAN
FUNCTION
. TRANSFORMATION
. . INSTRUMENTATION
. . EDITING
. . TRANSLATION
. . FORMATTING
. STATIC ANALYSIS
. . STRUCTURE CHECKING
. DYNAMIC ANALYSIS
. . COVERAGE ANALYSIS
. . TRACING
. . ASSERTION CHECKING
. . TUNING
OUTPUT
. USER OUTPUT
. . GRAPHICS
. . LISTINGS
. . DIAGNOSTICS
. MACHINE OUTPUT
. . SOURCE CODE OUTPUT
. . . FORTRAN

IMPLEMENTATION LANGUAGE: FORTRAN

TOOL PORTABLE: YES, TOOL SIZE: 25K WORKS

TOOL AVAILABLE: YES, PUBLIC DOMAIN: NO

RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): LICENSE FEE

TOOL SUPPORTED: YES, TOOL SUPPORT: GENERAL RESEARCH CORPORATION

CONTACT: WILLIAM R. DE HAAN, GENERAL RESEARCH CORP, 5383 HOLLISTER AVE, PO BOX 6770, SANTA BARBARA, CA, 93111, USA, 805-964-7724

RXVP80™

A SOFTWARE DOCUMENTATION, ANALYSIS, AND TEST SYSTEM

A SOFTWARE ENGINEERING TOOL DEMONSTRATION

Sabina H. Saib, Jeoffrey P. Benson,
Carolyn Gannon, and William R. DeHaan

1. Introduction

This proposal is in response to the call for software engineering tool developers to demonstrate their accomplishments at the 5th International Conference on Software Engineering in San Diego, March 9-12, 1981.

We will be demonstrating the RXVP80™ system, developed in The Software Workshop™ by General Research Corporation, Santa Barbara, California.

2. Summary of RXVP80™

RXVP80™ is a system of tools which perform a number of functions such as documentation, analysis, and test assistance, for code written in common dialects of FORTRAN including FORTRAN 77. The heart of the system is a library capable of storing the results of analysis of very large programs (>100,000 source lines). The system performs much of its analysis on an internal representation of the program as a directed graph. One of the primary features of RXVP80™ is its ability to analyze only the new or changed modules of a program, using the stored library to check interfaces.

Documentation features of RXVP80™ include calling trees, input/output reports, COMMON matrices, cross references, and statement type summaries. Analysis functions include type checking, parameters checking, set/use checking, and graph connectivity checking. Test assistance functions include path instrumentation, test coverage reports, and reaching set reports.

3. A Scenario of an RXVP80™ Demonstration

A demonstration of RXVP80™ consists of four parts. The first part is a demonstration of the STATIC ANALYSIS feature (see SAMPLE OUTPUT, Figure 1). The second part is a demonstration of the use of a Project Library to illustrate the checking of interface information (see SAMPLE OUTPUT, Figure 2). The third part is a demonstration of the DOCUMENTATION feature (see SAMPLE OUTPUT, Figure 3). The fourth part is a demonstration of the EXECUTION COVERAGE ANALYSIS (INSTRUMENTATION) feature (see SAMPLE OUTPUT, Figure 4).

The demonstration will take one hour. We will repeat it as often as necessary. We can handle about 10 people in each interactive demonstration.

4. RXVP80™ Literature

User's Manual

A description of the uses of RXVP80™ with command descriptions and sample outputs.

Collected Papers

A collection of articles written by members of the Software Quality Department.

5. Station, Day and Time

Forum, Tuesday, March 10, from 11:00 am until 12:30 pm.
Forum, Wednesday, March 11, from 9:00 am until 10:30 am.
Station 9, Thursday, March 12, from 9:00 am until 5:00 pm.

6. The Demonstrators

Sabina H. Saib is Director of the Software Quality Department at General Research. She received a BS and a PhD in Engineering from UCLA and an MSEE from the University of Maryland. She has published several papers on the verification and validation of software. Her current work is in the automated verification of embedded systems.

Carolyn Gannon is a member of the Software Quality Department at General Research. She received her BS and MS with a computer science specialty from the University of California, Santa Barbara. She is active in the JOVIAL users group and is leading the development of an automated verification system for JOVIAL J73. She has published several papers on test techniques and automated verification systems.

Jeoffrey P. Benson received his BA degree from California State University, Fresno, and his MSEE and PhD degrees from the University of California, Santa Barbara. Since 1973 Dr. Benson has been at General Research Corporation, where he has done research in software verification. He has also taught courses in compiler design and construction at the University of California, Santa Barbara. His current research interests include compiler-compilers and biomedical computing. Dr. Benson is a member of IEEE and ACM.

William R. DeHaan is Software Marketing Director at General Research. He received his BS from Fairleigh Dickinson University and is currently active in the creation, preparation, and marketing of proprietary software. Mr. DeHaan is a member of ACM.

```
STATIC ANALYSIS                    SUBROUTINE CIRCLE ( AREA )                    PAGE      2

 STMT NEST  LINE  SOURCE...                                                  ...SOURCE TAB
-------------------------------------------------------------------------------------------
    1          1      SUBROUTINE CIRCLE ( AREA )                                CIRCLE 2
    2          2      INTEGER AREA                                              CIRCLE 3
    3          3      DATA PI / 3.1416 /                                        CIRCLE 4
    4          4      RADIUS=DIAMTR/2                                           CIRCLE 5
                    ----------------------------------------------------------------------
                    -                          SET/USE ERROR                             -
                    - VARIABLE DIAMTR     USED BUT NEVER SET        REFER TO STATEMENT(S)-
                    -  4
                    ----------------------------------------------------------------------
    5          5      AREA=PI*RADIUS**2                                         CIRCLE 6
                    ----------------------------------------------------------------------
                    -                          MODE WARNING                              -
                    - LEFT HAND SIDE HAS MODE INTEGER   RIGHT HAND SIDE HAS MODE REAL    -
                    ----------------------------------------------------------------------
    6          6      IF (AREA.GT.50)                                           CIRCLE 7
    7    1     7      .   CALL PRINT (AREA)                                     CIRCLE 8
                    ----------------------------------------------------------------------
                    -                          MODE WARNING                              -
                    -PARAMETER  1 OF PRINT      ACTUAL PARAMETER HAS MODE INTEGER        -
                    -                           FORMAL PARAMETER HAS MODE REAL           -
                    ----------------------------------------------------------------------
                    ----------------------------------------------------------------------
                    -                          CALL ERROR                                -
                    -         PRINT      CALLED WITH  1 ACTUALLY HAS  2 ARGUMENTS        -
                    ----------------------------------------------------------------------
    8          8      END IF                                                    CIRCLE 9
    9          9      RETURN                                                    CIRCLE10
   10         10      CALL STACK (RADIUS,AREA)                                  CIRCLE11
                    ----------------------------------------------------------------------
                    -                          GRAPH WARNING                             -
                    -        STATEMENT  10  IS UNREACHABLE OR IS IN AN INFINITE LOOP     -
                    ----------------------------------------------------------------------
   11         11      END                                                       CIRCLE12
-------------------------------------------------------------------------------------------
                     STATIC   ANALYSIS   SUMMARY      ERRORS  WARNINGS
                     ---------------------------      ------  --------
                     GRAPH CHECKING                        0         1
                     CALL CHECKING                         1         0
                     MODE CHECKING                         0         2
                     SET/USE CHECKING                      1         0
          CALL CHECKING WAS NOT PERFORMED FOR THE FOLLOWING UNKNOWN EXTERNALS ...
          STACK
```

```
STATIC ANALYSIS                  SUBROUTINE PRINT (RADIUS,AREA)                  PAGE      3

 STMT NEST  LINE  SOURCE...                                                  ...SOURCE TAB
-------------------------------------------------------------------------------------------
    1          1      SUBROUTINE PRINT (RADIUS,AREA)                            PRINT  2
    2          2      WRITE (6,100) RADIUS, AREA                                PRINT  3
    3          3  100 FORMAT (12H FOR RADIUS ,F10.5,10H, AREA IS ,F10.5)        PRINT  4
    4          4      RETURN                                                    PRINT  5
    5          5      END                                                       PRINT  6
-------------------------------------------------------------------------------------------
                     STATIC   ANALYSIS   SUMMARY      ERRORS  WARNINGS
                     ---------------------------      ------  --------
                     GRAPH CHECKING                        0         0
                     CALL CHECKING                         0         0
                     MODE CHECKING                         0         0
                     SET/USE CHECKING                      0         0
```

FIGURE 1

RXVP80™

Sample STATIC ANALYSIS Report

```
STATIC ANALYSIS                    SUBROUTINE AVER ( A, N, ANS )                    PAGE      2

 STMT NEST  LINE  SOURCE...                                                        ...SOURCE TAB
-------------------------------------------------------------------------------------------------
    1          1      SUBROUTINE AVER ( A, N, ANS )
    2          2      INTEGER N, I, J
    3          3      REAL A(1), ANS, SUM
    4          4      J = 1
                    ------------------------------------------------------------------------
                    -                          SET/USE WARNING                             -
                    - VARIABLE J          SET BUT NEVER USED        REFER TO STATEMENT(S)-
                    -  4
                    ------------------------------------------------------------------------
    5          5      WHILE ( I .LE. N )
    6    1     6      .   SUM = SUM + A(I)
    7    1     7      .   I = I + 1
    8          8      END WHILE
    9          9      ANS = SUM/FLOAT(N)
   10         10      RETURN
   11         11      END
-------------------------------------------------------------------------------------------------
                     STATIC   ANALYSIS   SUMMARY      ERRORS  WARNINGS
                     ------------------------------   ------  --------
                    GRAPH CHECKING                        0         0
                    CALL CHECKING                         0         0
                    MODE CHECKING                         0         0
                    SET/USE CHECKING                      0         1
          CALL CHECKING WAS NOT PERFORMED FOR THE FOLLOWING UNKNOWN EXTERNALS ...
          FLOAT

...WRITING INTERFACE LIBRARY

...  1898 WORDS WRITTEN
FORTRAN STOP
RXVP80 processing has ended.
```

FIGURE 2(a)

RXVP80™

PROJECT LIBRARY USAGE

Creating a Project Library File

With One Subprogram

```
 G R C  SOFTWARE VERIFICATION PROGRAM -- RXVP80

ENTER COMMANDS FOLLOWED BY END-OF-FILE

EXPAND.
OPTION=STATIC,LIST.

...READING INTERFACE LIBRARY

...END OF LIBRARY ENCOUNTERED

STATIC ANALYSIS                    PROGRAM STAT                                     PAGE      2

 STMT NEST  LINE  SOURCE...                                                        ...SOURCE TAB
-------------------------------------------------------------------------------------------------
    1          1      PROGRAM STAT
    2          2      INTEGER DATA(10)
    3          3      REAL RESULT
    4          4      READ *, DATA
    5          5      CALL AVER ( DATA, RESULT )
                    ------------------------------------------------------------------------
                    -                          MODE WARNING                                -
                    -PARAMETER  1 OF AVER   ACTUAL PARAMETER HAS MODE INTEGER             -
                    -                       FORMAL PARAMETER HAS MODE REAL               -
                    ------------------------------------------------------------------------
                    ------------------------------------------------------------------------
                    -                          MODE WARNING                                -
                    -PARAMETER  2 OF AVER   ACTUAL PARAMETER HAS MODE REAL                -
                    -                       FORMAL PARAMETER HAS MODE INTEGER            -
                    ------------------------------------------------------------------------
                    ------------------------------------------------------------------------
                    -                          CALL ERROR                                  -
                    -         AVER       CALLED WITH  2 ACTUALLY HAS  3 ARGUMENTS         -
                    ------------------------------------------------------------------------
    6          6      PRINT *, RESULT
    7          7      STOP
    8          8      END
-------------------------------------------------------------------------------------------------
                     STATIC   ANALYSIS   SUMMARY      ERRORS  WARNINGS
                     ------------------------------   ------  --------
                    GRAPH CHECKING                        0         0
                    CALL CHECKING                         1         0
                    MODE CHECKING                         0         2
                    SET/USE CHECKING                      0         0
```

FIGURE 2(b)

RXVVP80™

PROJECT LIBRARY FILE

Using the Project Library File

Created in Figure 2(a) by Adding a Subprogram

STATEMENT PROFILE SUBROUTINE SUBA (K1,K2,K3,K4,N) PAGE 27

INTERFACE CHARACTERISTICS

ARGUMENTS	5
COMMON	1
ENTRY	1
EXIT	2
WRITE	12

STATEMENT CLASSIFICATION	STATEMENT TYPE	NUMBER	PERCENT
DECLARATION...			
	DATA	3	4.8
	COMMON	1	1.6
	FORMAT	2	3.2
	TOTAL	6	9.5
EXECUTABLE...			
	DO	1	1.6
	END	1	1.6
	ASSIGNMENT	11	17.5
	GOTO	7	11.1
	SUBROUTINE	1	1.6
	WRITE	12	19.0
	CONTINUE	4	6.3
	RETURN	2	3.2
	TOTAL	39	61.9
DECISION...			
	IF	4	6.3
	CONTINUE	1	1.6
	TOTAL	5	7.9
DOCUMENTATION...			
	COMMENT	13	20.6
	TOTAL	13	20.6

FIGURE 3(f)

RXVP80™

Sample Local STATEMENT PROFILE Report

INVOCATION SUMMARY PAGE 28

ENTRY	LISTS OF CALLS
BKDEMO	IS CALLED BY - -NONE-
	AND CALLS - -NONE-
MAIN	IS CALLED BY - -NONE-
	AND CALLS - SUBA
SUBA	IS CALLED BY - MAIN
	AND CALLS - -NONE-

FIGURE 3(g)

RXVP80™

Sample Global INVOCATION SUMMARY Report

COMMON SUMMARY PAGE 29

COMMON	MODULES WHICH INCLUDE THE COMMON
NAMES	BKDEMO MAIN SUBA

FIGURE 3(h)

RXVP80™

Sample Global COMMON SUMMARY Report

```
COMMON MATRICES                                                                                    PAGE     30

LEGEND (C=FIRST USED IN A CALL,E=EQUIVALENCED,S=SET,U=USED,X=SET AND USED)
-------------------------------------------------------------------------------------------------------------
-----------------------------
       **          *       *
       * * MODULE  * B M S *
       *  *        * K A U *
       *   *       * D I B *
       *    *      * E N A *
       *     *     * M     *
       *      *    * O     *
       *       *   *       *
 COMMON * SYMBOL  * *       *
       *          **        *
-----------------------------
NAMES   * ISUBS     * S U U *
-----------------------------
```

FIGURE 3(I)

Sample Global COMMON MATRICES Report

```
I/O STATEMENTS                                                                                     PAGE     31


THE FOLLOWING MODULES CONTAIN I/O STATEMENTS

              MAIN
              SUBA

-------------------------------------------------------------------------------------------------------------------------------------------

I/O STATEMENTS AND ASSOCIATED FORMATS
-------------------------------------

--- MAIN    ---

 STMT NEST  LINE  SOURCE...                                                                                                  ...SOURCE TAB
-------------------------------------------------------------------------------------------------------------------------------------------
    8          15        WRITE (6,600)
    9          16   600  FORMAT (50H1EXECUTION COVERAGE (INSTRUMENTATION) TEST PROGRAM,//)
               17  C
   10          18        WRITE (6,599) ISUBS(1)
   11          19   599  FORMAT (4H IN ,A4)
   13          21        WRITE (6,602) L1,IA1
   14          22   602  FORMAT (1H ,5X,8HPATH IS ,A4,1H,,I5,7H TIMES.)
   16          25        READ (5,500) KSUB,K1,K2,K3,K4,N
   17          26   500  FORMAT (5(1X,A4),I3,52X)
   18          27        WRITE (6,601) KSUB,K1,K2,K3,K4,N
   19          28   601  FORMAT (1H ,16HINPUT DATA IS - ,5(A4,1H,),I3,1H.)
               29  C
   23          32        WRITE (6,602) L3,IA3
   24          33        WRITE (6,605) KASE
   25          34   605  FORMAT (29H ********** END OF TEST CASE ,I3,11H ********** )
   30          40        WRITE (6,602) L2,IA2
               41  C
   34          44        WRITE (6,602) L4,IA4
   35          45        WRITE (6,603)
   36          46   603  FORMAT (43HO---------- PROCESSING COMPLETED ----------)
   40          51        WRITE (6,602) L5,IA5
               52  C
   44          55        WRITE (6,602) L11,IA11
   45          56        WRITE (6,604)
   46          57   604  FORMAT ( 1H ,21X,16H(NOT RECOGNIZED))
   49          61        WRITE (6,602) L6,IA6

--- SUBA    ---

 STMT NEST  LINE  SOURCE...                                                                                                  ...SOURCE TAB
-------------------------------------------------------------------------------------------------------------------------------------------
    6           9        WRITE (6,599) ISUBS(2)
```

FIGURE 3(J)

RXVP80™

Sample Global I/O STATEMENTS Report

```
COMMON CROSS REFERENCE                                                                             PAGE     33

 NAME      SCOPE     MODULE    USED/SET/EQUIVALENCED  ( * INDICATES SET )
-------------------------------------------------------------------------------------------------------------
ISUBS      NAMES     BKDEMO      3*
           NAMES     MAIN       10
           NAMES     SUBA        6
-------------------------------------------------------------------------------------------------------------
```

FIGURE 3(k)

RXVP80™

Sample Global COMMON CROSS REFERENCE Report

```
DD-PATH DEFINITIONS                    SUBROUTINE SUBA ( K1,K2,K3,K4,N )                    PAGE    19

 STMT NEST  LINE  SOURCE....                                                                ...SOURCE TAB
-------------------------------------------------------------------------------------------------------------
    1          1        SUBROUTINE SUBA ( K1,K2,K3,K4,N )
               2  C
                                                                  ** DDPATH   1 IS PROCEDURE ENTRY
    2          3        COMMON / NAMES / ISUBS(4)
               4  C
    3          5        DATA L1,L3,L5,L7,L9,L11 / 2HA1,2HA3,2HA5,2HA7,2HA9,3HA11 /
    4          6        DATA L2,L4,L6,L8,L10 / 2HA2,2HA4,2HA6,2HA8,3HA10 /
    5          7        DATA IA1,IA2,IA3,IA4,IA5,IA6,IA7,IA8,IA9,IA10,IA11 / 11*0 /
               8  C
    6          9        WRITE (6,599) ISUBS(2)
    7         10   599  FORMAT (1HO,10X,3HIN ,A4)
    8         11        IA1=IA1+1
    9         12        WRITE (6,600) L1,IA1
   10         13   600  FORMAT (1H ,15X,8HPATH IS ,A4,1H,,I5,7H TIMES.)
              14  C
   11         15        IF ( K1.EQ.L2 ) GO TO 100
                                                                  ** DDPATH   2 IS TRUE BRANCH
                                                                  ** DDPATH   3 IS FALSE BRANCH
   13         16        IA3=IA3+1
   14         17        WRITE (6,600) L3,IA3
   15         18        GO TO 110
              19  C
   16         20   100  CONTINUE
   17         21        IA2=IA2+1
   18         22        WRITE (6,600) L2,IA2
              23  C
   19         24   110  IF ( K2.EQ.L4 ) GO TO 200
                                                                  ** DDPATH   4 IS TRUE BRANCH
                                                                  ** DDPATH   5 IS FALSE BRANCH
   21         25        IA5=IA5+1
   22         26        WRITE (6,600) L5,IA5
   23         27        GO TO 210
              28  C
   24         29   200  CONTINUE
   25         30        IA4=IA4+1
   26         31        WRITE (6,600) L4,IA4
              32  C
   27         33   210  IF ( K3.EQ.L6 ) GO TO 300
                                                                  ** DDPATH   6 IS TRUE BRANCH
                                                                  ** DDPATH   7 IS FALSE BRANCH
   29         34        IA7=IA7+1
   30         35        WRITE (6,600) L7,IA7
   31         36        GO TO 310
              37  C
   32         38   300  CONTINUE
   33         39        IA6=IA6+1
   34         40        WRITE (6,600) L6,IA6
              41  C
-------------------------------------------------------------------------------------------------------------
```

FIGURE 4(a)

RXVP80™

Sample DD-PATH DEFINITIONS Report

Showing Where Probes Were Automatically Placed

```
RECORD OF DECISION TO DECISION (DD PATH) EXECUTION

-------------------------------------------------------------------------------------------------------------

MODULE  $SUBA   $          TEST CASE NO.        1

-------------------------------------------------------------------------------------------------------------
DD PATH I  NO.  NOT EXECUTED  I   NUMBER OF EXECUTIONS -- NORMALIZED TO MAXIMUM        I         I  NUMBER OF
NUMBER  I                     I.------20.-------40.-------60.-------80.------100.     I         I EXECUTIONS
-------------------------------------------------------------------------------------------------------------
        I                     I
    1   I                     I XXXXX                                                  I    1    I        6
    2   I   2          00000  I                                                        I         I
    3   I                     I XXXXX                                                  I    3    I        6
    4   I                     I XX                                                     I    4    I        3
    5   I                     I XX                                                     I    5    I        3
    6   I                     I X                                                      I    6    I        2
    7   I                     I XXX                                                    I    7    I        4
    8   I                     I                                                        I    8    I        1
    9   I                     I XXXX                                                   I    9    I        5
   10   I                     I XXXX                                                   I   10    I        5
   11   I                     I XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   I   11    I       57
-------------------------------------------------------------------------------------------------------------

                                                          TOTAL NUMBER OF DD PATH EXECUTIONS =      92

   TOTAL OF  1  NOT EXECUTED     EXECUTED  10/ 11       PERCENT EXECUTED =    90.91
```

FIGURE 4(b)

Sample DETAILED EXECUTION COVERAGE Report

For Program in Figure 4(a)

		SUMMARY -- THIS TEST				CUMULATIVE SUMMARY			
TEST CASE	MODULE NAME	NUMBER OF D-D PATHS	NUMBER OF INVOCATIONS	D-D PATHS TRAVERSED	PER CENT COVERAGE	NUMBER OF TESTS	INVOCATIONS	TRAVERSED	COVERAGE
1	MAIN	7	1	5	71.43	1	1	5	71.43
	SUBA	11	6	10	90.91	1	6	10	90.91
	$$ALL$$	18		15	83.33	1		15	83.33
2	MAIN	7	0	4	57.14	2	1	6	85.71
	SUBA	11	2	11	100.00	2	8	11	100.00
	$$ALL$$	18		15	83.33	2		17	94.44

FIGURE 4(c)

RXVP80™

Sample SUMMARY EXECUTION COVERAGE Report

MODULE NAME	TEST NUMBER	PATHS NOT HIT	LIST OF DECISION TO DECISION PATHS NOT EXECUTED
<MAIN >	2	3	1 3 7
	CUMUL	1	7
<SUBA >	2	0	
	CUMUL	0	

FIGURE 4(d)

RXVP80™

Sample "NOT HIT" EXECUTION COVERAGE Report

FEATURE CLASSIFICATION:

INPUT
. SUBJECT
. . CODE INPUT
. . . FORTRAN
. . . IFTRAN
FUNCTION
. TRANSFORMATION
. . INSTRUMENTATION
. . TRANSLATION
. . FORMATTING
. . RESTRUCTURING
. STATIC ANALYSIS
. . MANAGEMENT
. . ERROR CHECKING
. . DATA FLOW ANALYSIS
. . STRUCTURE CHECKING
. . CROSS REFERENCE
. . STATISTICAL ANALYSIS
. . CONSISTENCY CHECKING
. . TYPE ANALYSIS
. . COMPLEXITY MEASUREMENT
. . COMPLETENESS CHECKING
. . SCANNING
. . INTERFACE ANALYSIS
. . UNITS ANALYSIS
. DYNAMIC ANALYSIS
. . COVERAGE ANALYSIS
. . TRACING
. . ASSERTION CHECKING
. . TUNING
. . CONSTRAINT ANALYSIS
. . SYMBOLIC EXECUTION
OUTPUT
. USER OUTPUT
. . GRAPHICS
. . TABLES
. . LISTINGS
. . DIAGNOSTICS
. MACHINE OUTPUT
. . SOURCE CODE OUTPUT
. . . FORTRAN
. . . IFTRAN

IMPLEMENTATION LANGUAGE: FORTRAN, TOOL PORTABLE: YES, TOOL SIZE: 5ØK WORDS, TOOL AVAILABLE: YES, PUBLIC DOMAIN: NO, RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.): LICENSE, TOOL SUPPORTED: YES, TOOL SUPPORT: GENERAL RESEARCH CORP.

CONTACT: WILLIAM R. DEHAAN, GENERAL RESEARCH CORP, 5383 HOLLISTER AVE, PO BOX 677Ø, SANTA BARBARA, CA, 93111, USA, 8Ø5-964-7724

APPENDIX

FEATURES OF THE SAN DIEGO TOOL FAIR

The following is a cross reference to the features of the tools that were demonstrated at the San Diego Tool Fair. The cross reference provides a quick way to identify tools of interest. This cross reference was developed from the feature classifications that appear at the end of each tool summary.

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBS SP 500-80	2. Performing Organ. Report No.	3. Publication Date October 1981

4. TITLE AND SUBTITLE

Proceedings of the NBS/IEEE/ACM Software Tool Fair
Held in conjunction with the 5th International Conference on Software Engineering in San Diego, CA, March 10-12, 1981

5. AUTHOR(S)

Raymond C. Houghton, Jr., Editor

6. PERFORMING ORGANIZATION *(If joint or other than NBS, see instructions)*

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS *(Street, City, State, ZIP)*

IEEE Computer Society
P.O. Box 639
Silver Spring, MD 20901

and

SIGSOFT ACM
1133 Avenue of the Americas
New York, NY 10036

10. SUPPLEMENTARY NOTES Library of Congress Catalog Card Number: 81-600109

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)*

This document summarizes the presentations made by each demonstrator at the San Diego Tool Fair. The San Diego Tool Fair was a first-of-its-kind demonstration of software engineering tools at a major conference. Each summary includes a short description of the tool, a scenario of the demonstration, a list of references, background on the demonstrators, sample output, and a page of miscellaneous data obtained from the NBS Software Tools Database. The appendix provides a cross reference to the features of the tools.

12. KEY WORDS *(Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)*

programming aids; software automation; software development; software engineering; software testing; software tools.

13. AVAILABILITY

☒ Unlimited
☐ For Official Distribution. Do Not Release to NTIS
☒ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
☐ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

238

15. Price

$6.50

USCOMM-DC 6043-P80

☆U.S. GOVERNMENT PRINTING OFFICE: 1981-340-997/1789

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SCIENCE & TECHNOLOGY

Superintendent of Documents,
Government Printing Office,
Washington, D. C. 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Bureau of Standards Special Publication 500-

Name _____

Company _____

Address _____

City _____ State _____ Zip Code _____

(Notification key N-503)

NBS TECHNICAL PUBLICATIONS

PERIODICALS

JOURNAL OF RESEARCH—The Journal of Research of the National Bureau of Standards reports NBS research and development in those disciplines of the physical and engineering sciences in which the Bureau is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Bureau's technical and scientific programs. As a special service to subscribers each issue contains complete citations to all recent Bureau publications in both NBS and non-NBS media. Issued six times a year. Annual subscription: domestic $13; foreign $16.25. Single copy, $3 domestic; $3.75 foreign.

NOTE: The Journal was formerly published in two sections: Section A "Physics and Chemistry" and Section B "Mathematical Sciences."

DIMENSIONS/NBS—This monthly magazine is published to inform scientists, engineers, business and industry leaders, teachers, students, and consumers of the latest advances in science and technology, with primary emphasis on work at NBS. The magazine highlights and reviews such issues as energy research, fire protection, building technology, metric conversion, pollution abatement, health and safety, and consumer product performance. In addition, it reports the results of Bureau programs in measurement standards and techniques, properties of matter and materials, engineering standards and services, instrumentation, and automatic data processing. Annual subscription: domestic $11; foreign $13.75.

NONPERIODICALS

Monographs—Major contributions to the technical literature on various subjects related to the Bureau's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NBS, NBS annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NBS under the authority of the National Standard Data Act (Public Law 90-396).

NOTE: The principal publication outlet for the foregoing data is the Journal of Physical and Chemical Reference Data (JPCRD) published quarterly for NBS by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements available from ACS, 1155 Sixteenth St., NW, Washington, DC 20056.

Building Science Series—Disseminates technical information developed at the Bureau on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NBS under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NBS administers this program as a supplement to the activities of the private sector standardizing organizations.

Consumer Information Series—Practical information, based on NBS research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

Order the **above** *NBS publications from: Superintendent of Documents, Government Printing Office, Washington, DC 20402.*

Order the **following** *NBS publications—FIPS and NBSIR's—from the National Technical Information Services, Springfield, VA 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NBS pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NBS Interagency Reports (NBSIR)—A special series of interim or final reports on work performed by NBS for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Services, Springfield, VA 22161, in paper copy or microfiche form.

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Washington, D.C. 20234

OFFICIAL BUSINESS

Penalty for Private Use, $300

POSTAGE AND FEES PAID
U.S. DEPARTMENT OF COMMERCE
COM-215

SPECIAL FOURTH-CLASS RATE
BOOK