NBS Special Publication 500-80
Proceedings of the NBS/IEEE/ACM Software Tool Fair
The National Bureau of Standards 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
- 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
- Mechanical Engineering and Process Technology

**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.

---

1Headquarters and Laboratories at Gaithersburg, MD, unless otherwise noted; mailing address Washington, DC 20234.
2Some 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
CODEN: XNBSAV

Library of Congress Catalog Card Number: 81-600109
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.

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

CS4: A Tool for the Design and Generation of Database Systems ........................................ 1
S. Berild
University of Stockholm, Stockholm, Sweden
S. Nachmens
Databaskonsult DBK AB, Solna, Sweden

Station 2

LILITH: A Modula Machine ........................................ 7
N. Wirth
Institut fuer Informatik, ETH, Zurich, Switzerland
R. Ohran
 Brigham Young University, Provo, UT

Station 3

SARA: A Demonstration of the Development of Interactive Software with Integral Help .............. 10
Robert S. Fenchel
University of California, Los Angeles, CA

Station 4

VIRTUAL OS: The Software Tools Virtual Operating System Project ........................................ 15
Debbie Scherrer
Lawrence Berkeley Laboratory, Berkeley, CA
Dave Martin
Hughes Aircraft Company, Los Angeles, CA
Chris Peterson
Orincon Corp., La Jolla, CA

Station 5

SCG: Structure Chart Graphics System ......................... 21
Debra L. Resendez and James W. Winchester
Hughes Aircraft Company, Fullerton, CA

DQM: Design Quality Metrics System ......................... 25
Debra L. Resendez and James W. Winchester
Hughes Aircraft Company, Fullerton, CA
AISIM: Automated Interactive Simulation Modelling System ........................................ 29
William P. Austell, Jr. and Ronald R. Willis
Hughes Aircraft Company, Fullerton, CA

Station 6

THE ENGINE: COBOL Structuring Engine ................................. 33
Jon Cris Miller and Michael J. Lyons
Catalyst Corporation, La Grange, IL

Station 7

PSL/PSA: Problem Statement Language/Problem Statement Analyzer .......................... 38
E. Callender
Jet Propulsion Laboratory, Pasadena, CA
Hasan H. Sayani
Advanced Systems Technology, Greenbelt, MD
Daniel Teichroew
University of Michigan, Ann Arbor, MI

SDDL: Software Design and Documentation Language ............ 44
E. Callender, C. Hartsough, and H. Kleine
Jet Propulsion Laboratory, Pasadena, CA

Station 8

SLIM: A Quantitative Tool for Software Cost and Schedule Estimation ........................ 49
Lawrence H. Putnam
Quantitative Software Management, Inc., Mc Lean, VA

Station 9

POD: Performance Oriented Design ................................. 58
A. Levy
BGS Systems, Inc., Waltham, MA

Station 11

PWB FOR VAX/VMS: Programmer Workbench Tools on VAX/VMS ............................... 64
Heinz Lycklama
Interactive Systems Corporation, Santa Monica, CA
March 11, 1981

Station 1

AFFIRM: A Specification and Verification System .......... 69
R. Erickson, S. Gerhart, S. Lee, and D. Thompson
USC/Information Sciences Institute,
Marina Del Ray, CA

Station 3

SARA: SARA as a Tool for Software Design:
Building-block Modelling and Composition ............... 76
Maria Heloisa Penedo
University of California, Los Angeles, CA

Station 4

ARGUS/MICRO: ARGUS in the Microcomputer Environment .... 83
William C. King
Boeing Computer Services Company, Seattle, WA

DYNA: A Tool From the ARGUS Toolbox ....................... 88
Leon G. Stucki
Boeing Computer Services Company, Seattle, WA

COMMAP: A Tool from the ARGUS Toolbox ....................... 96
Leon G. Stucki
Boeing Computer Services Company, Seattle, WA

Station 5

LOGICFLOW: A Software Design and Analysis Tool .......... 103
S. Moy, G. Nielsen, R. Sanchez, T. Wallace,
and S. Mc Wethy
Logicon, San Pedro, CA

Station 6

SREM: Software Requirements Engineering Methodology .... 111
R. H. Hoffman, R. P. Loshbough, and R. W. Smith
TRW Inc., Huntsville, AL

Station 7

SDP: A Computerized Tool for System Design
and Maintenance .............................................. 117
Nancy Linden and Moshe Yavne
Mayda Software Engineering, Rehovot, Israel

viii
SOFTOOL 80 (TM)*: A Methodology and a Comprehensive Set of Tools for Software Management, Development, and Maintenance ........................................ 122
Richard Hug and Thomas Strelitch
Softool Corporation, Goleta, CA
Michael Resnicow and Robert Ahola
Systems Engineering Laboratories, Fort Lauderdale, FL

Station 9

ITB: Interactive Test Bed ................................. 128
James B. Henderson and Edward F. Miller, Jr.
Software Research Associates, San Francisco, CA

ISUS: Interactive Semantic Update System ............... 136
James B. Henderson and Edward F. Miller, Jr.
Software Research Associates, San Francisco, CA
Morton Hirschberg
U. S. Army Ballistic Research Laboratory,
Aberdeen Proving Ground, MD

Station 11

FAME: Front-End Analysis and Modeling Environment ...... 146
J. Rosenbaum and C. Early
Higher Order Software, Inc., Jericho, NY

March 12, 1981

Station 1

Schemacode: An Interactive Schematic Pseudocode for Program Development, Documentation, and Structured Coding ........................................ 152
Pierre N. Robillard and Rejean Plamondon
Ecole Polytechnique de Montreal, Montreal, Canada

* SOFTOOL 80 is a Trade Mark of Softool Corporation
Station 2

**INSTRU:** An Automated Software Instrumentation System ..... 159
J. C. Huang
University of Houston, Houston, TX

Station 3

**SARA:** Control-Flow Analysis in SARA ...................... 163
Rami R. Razouk
University of California, Los Angeles, CA

Station 4

**ONLINE ASSIST:** A User Interface for Online Assistance .... 167
Nathan Relles
Sperry Univac, Blue Bell, PA
Lynne A. Price
BNR, Inc., Mountain View, CA

Station 5

**FTN-77 ANALYZER:** FORTRAN 77 Analyzer ...................... 174
John Barkley and Patricia Powell
National Bureau of Standards, Gaithersburg, MD

**TOOLS DATABASE:** NBS Software Tools Database ............. 180
Raymond C. Houghton, Jr. and Karen A. Oakley
National Bureau of Standards, Gaithersburg, MD

Station 6

**SRIMP:** Software Requirements Integrated Modeling
Program ....................................................... 186
Stephanie White
Grumman Aerospace Corp., Bethpage, NY

Station 7

**AUTO-DBO:** Automated Design by Objectives ................. 190
Tom Gilb
Independent EDP Consultant, Kolbotn, Norway
Lech Krzanik
University of Krakow, Krakow, Poland
Stations 8 & 10

UCSD P-SYSTEM: A Portable Software Development System .... 194
Mark Overgaard and Joan Gianetta
Softech Microsystems, San Diego, CA

MSEF: The Microprocessor Software Engineering Facility ... 197
Rich Thall
SofTech, Inc., Waltham, MA
Gail Anderson
Softech Microsystems, San Diego, CA

Station 9

IFTRAN (TM)*: A Preprocessor for FORTRAN ............... 200
Sabina H. Saib, Jeoffrey P. Benson, Carolyn Gannon,
and William R. DeHaan
General Research Corp., Santa Barbara, CA

RXVP80 (TM)*: A Software Documentation, Analysis, and
Test System .......................................................... 208
Sabina H. Saib, Jeoffrey P. Benson, Carolyn Gannon,
and William R. DeHaan
General Research Corp., Santa Barbara, CA

* 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
Huvudstugatan 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 programming 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.


5. THE DEMONSTRATORS

Stig Berild

Stig Berild has been a researcher at the Department of Computer Science, University of Stockholm, since 1970. He is currently a member of the Systems 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

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 preceding 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' SPACE 'SALARY', SPACE
50 FOREACH EMP IN ('EMPLOYEE', 'EXCO', =) DO
60 GET SAL IN ('SALARY', EMP, =)
70 OUTPUT LINE
80 NEXT
90 END
```
CS4

SAMPLE OUTPUT

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

<table>
<thead>
<tr>
<th>R CS4</th>
<th>Initiate CS4. Start of session</th>
</tr>
</thead>
<tbody>
<tr>
<td>* ASSIGN DB EXBAS</td>
<td>Assign a database</td>
</tr>
<tr>
<td>* ASSIGN LIB EXLIB</td>
<td>Assign a procedure library</td>
</tr>
<tr>
<td>* INSERT</td>
<td>Command for automatic generation of line numbers</td>
</tr>
<tr>
<td>10 BEGIN</td>
<td>Input of procedure</td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>90 END</td>
<td></td>
</tr>
<tr>
<td>* NAME SALARYLIST</td>
<td>Assign name to procedure</td>
</tr>
<tr>
<td>* SAVE</td>
<td>Save procedure in library</td>
</tr>
<tr>
<td>* RUN SALARYLIST</td>
<td>Execute procedure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANDERSON</td>
<td>5000</td>
</tr>
<tr>
<td>ERICSON</td>
<td>5600</td>
</tr>
<tr>
<td>JONES</td>
<td>4000</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>SMITH</td>
<td>6000</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>ZIMMERMAN</td>
<td>11500</td>
</tr>
</tbody>
</table>

The salary list is printed

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 achieve 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-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 (System Architect's Apprentice) 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-AF03-76SF0034 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


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.
MULTICS 34,30: MIT, Cambridge, Mass.
Load x 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.
SARA Fenchel logged in 11/25/80 2342.8 est Tue from TELNET ASCII
Last login 11/25/80 2340.2 est Tue from TELNET ASCII terminal "NET".
Time till:
You have no mail.
**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 contains
errors recovery information and a specification of the meaning of each language construct.

```
ac grammar
SLR(1) Grammar Preprocessor December 19, 1973
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 grammer >-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 N News-Command 5 */
```

The specification of the grammar (interaction language) is a RNF-like notation for a context
free language. Notice that several additional commands have been added. The D command
provides a semantic description. This description is used to generate interactive assistance
as well as user messages. E and N commands are optional. The information associated with
errors recovery and other productions of general use are also included. For example, the
combination of N and D commands is used to specify semantic rules such as
```
/* 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 Command-Terminator
Command-Terminator: N X X
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-NewsCommand | End-Command
D 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 news tool and return to the SARA selector
Display-Menu-Command: B Menu Date-option XL 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 or after the given date be displayed.
E : B Menu 4/5/79:
```

**Print-News-Command:** : B Menu News-Command XI 2%
D ! The print news command allows the user to view the
contents of the indicated news item.
N News-Command must be one of the items
available in the news system
(see Display-Menu-Command).
E ! B Menu help
br
End-Command: : B Menu News-Command XI 3%
D ! The end command ends the SARA news tool.
E ! B Menu tool;
Date-option: Date XI 1%
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 Year-Option
D ! A date is a calendar date where the year specified
is optional.
E ! 13/15/70
. br
9/21:
Year-option: Delimiter Year
D ! A year option is an optional specification of a year.
Month: integer XI 1%
D ); A month is a numerical (integer) specification
of a month of the year.
Time: integer XI 1%
D ); A day is a numerical (integer) specification of
a day of the month.
Day: integer XI 1%
D ); The day integer must be valid for whatever month
is being used.
Year: integer XI 1%
D ); A year is a numerical (integer) specification of
a calendar year. The year must be specified by the
last 2 digits of the calendar year only.
E 1786
br
82 1;
Delimiter: ;
D IA delimiter is a symbol used to separate the various
components of a date.
News-item: id XI 4%
qualified-id XI 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: N X X
News-Command: new-line X X X
D IA 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 = B Menu 8m
Sprint = B Menu 8p
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 terminal. A detailed listing of the
results of the grammar program is included in a file. If the processing completes
successfully, the grammar postprocessor will generate several files which will include parse
trees, help informations and user manual source.

Listing will appear in file: [pd]news.listing
Intermediate grammar processing started.
**the grammar is srl(1)**
Intermediate grammar processing completed.

Grammar Postprocessor
Working library now user=dir=dir>SARA>Fenchel>news
news grammar is srl(1)
End of Grammar Postprocessor
Converting parse tables to SARA format.
End of SLR(1) Grammar Preprocessor
**time 09:37 cost $1.09 $1.10**
The grammar processor has been completed successfully. Now the user tests the language
which has been constructed by using the input/output stream, user, parser, etc.
Results presented in the next section are based on the development environment.
The results are then the output of a system which uses semantic routines.
Thus, the output will be recognized, however no semantic processing will take place.
Notice that semantic and syntactic tools are available at this stage
as are error messages and error recovery. Also note that assistance may be included
via the F key at any point of system use. This is indicated by the F appearing both
preceding and following commands in this example.

The following are examples of parsed and syntactic error information. Notice the
use of 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
+ time 00:12 cost $0.17 $1.27

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


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 ....................................... uncompressed 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
suspend ................................ 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 handout. 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


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 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 lifecycle. 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 PROPRIETARY

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 hierarchy.
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


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 K. 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.

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:

5. Station, Day, and Time

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

6. The Demonstrators

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 systems. 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's structured 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. Output 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


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
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
- Eliminates perform thru overlap
- Reduces go to
- 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

\[\text{GO TO} \ldots \text{DEPENDING WITH NO EXPLICIT FALL THRU}\]
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

NAME=customers-and-vendors

+----INTF----+ +----INPUT----+ +----PROCESS----+ +----OUTPUT----+ +----INTF----+
Icustomers- 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 I See Index.
+------------+ +GENERATED+ +RECEIVES+ +GENERATED+ +RECEIVES+

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
NAME=customers

+  
    +---INPUT---  +---PROCESS---
    I orders- I Iorder- I User LINK

    +---INTF---  . I customers I Iprocessing I Limit of 2
    I I I . +GENERATED+ +RECEIVES--

I customers I...

I I I . +---INPUT---  +---PROCESS---

    +---INTF---  . I payments I I I User LINK
    I I I . +GENERATED+ +RECEIVES--

    +---INTF---  . I accounts I I I Limit of 2
    I customers I I I I accounting I

    +---INTF---  +---RECEIVED---  +---GENERATES---

PSL/PSA Demonstration - San Diego

NAME=customers

+  
    +---INTF---  +---OUTPUT---  +---PROCESS---
    I I I I bills- I I I User LINK

I customers I.....Ito- I.....Iaccounting I Limit of 2

I I I I customers I I I I reached

+---INTF---  +---RECEIVED---  +---GENERATES---

PSL/PSA Demonstration - San Diego
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 TEICHREROW, 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
SDDL Structures Illustration

Line | Program to Exemplify the SDDL Keywords
--- | ---
1 | PROGRAM TO EXEMPLIFY THE SDDL KEYWORDS
2 | ********************************************************************
3 | * THIS EXAMPLE ILLUSTRATES THE PROCESSOR'S RESPONSE TO KEYWORD *
4 | * STATEMENTS. *
5 | ********************************************************************
6 | LOOP UNTIL FINISHED
7 | IF PROPOSITION IS TRUE
8 | DO STEP.ONE (DEFERRED ABSTRACTION)---( )
9 | ELSE
10 | DO STEP.TWO-------------------------( )
11 | ENDIF
12 | IF THINGS LOOK BAD AT THIS POINT
13 | ------EXITLOOP
14 | ELSEIF THINGS LOOK DISASTROUS
15 | ------EXITPROGRAM
16 | ENDIF
17 | REPEAT UNTIL WE GET IT RIGHT
18 | CALL EXEMPLIFY TO DEMONSTRATE PAGE NUMBERING------( )
19 | ENDPROGRAM
SPECIFICATION FOR VEEP_FUNCTIONAL.REQUIREMENTS

THIS IS A BREAKDOWN OF THE FUNCTIONAL REQUIREMENTS OF THE VEHICLE.
ECONOMY AND EMISSIONS PERFORMANCE SIMULATION PROGRAMS.
EXPANSION OF THE ITEMS LISTED BELOW MAY BE FOUND ON THE PAGES WHOSE
NUMBERS ARE SHOWN ON THE RIGHT.

REQUIREMENT: SYNTAX_ANALYZER_AND_DICTIONARY_MANAGER

+ KEYWORDS FOR USER INTERACTIVE CONTROL OF THE MODEL

REQUIREMENT: DATA_BASE_MANAGEMENT_SYSTEM

+ VEHICLE COMPONENTS
+ DRIVING CYCLE
+ ENVIRONMENT

REQUIREMENT: REPORT_GENERATOR

REQUIREMENT: ENVIRONMENT_SIMULATOR

REQUIREMENT: VEHICLE_SIMULATOR

+ STEADY STATE OPERATION
+ MAXIMUM PERFORMANCE
+ DRIVING CYCLE PERFORMANCE

END_SPECIFICATION
Line 84: ENTITY 'ENGINE:
"NAME"
"TEXTUAL.DESCRIPTION"
"MAX.HORSEPOWER"
"DISPLACEMENT"
"MOMENT.OF.INERTIA"
"MAX.RPM"
"IDLE.RPM"
"IDLE.FUEL.FLOW"
"OUTPUT.REDUCTION.GEAR"
"STEADY.STATE.TEMP"
"TEST.FUEL.DENSITY"
"TEST.FUEL.HEAT.VALUE"
"TEST.DRY.BULB.TEMP"
"TEST.RELATIVE.HUMIDITY"
"TEST.BAROMETRIC.PRESSURE"
"MAX.TORQUE.BOUNDARY" (VECTOR)
"MIN.TORQUE.BOUNDARY" (VECTOR)
"MAX.TORQUE"
"MAX.TORQUE.RPM"

Line 105: SET 'DYNOMAP' RANKED BY LOW RPM VALUE
"NO.OF.VALUES" (AT EACH POINT OF THE DYNOMAP)
"VALUE.NAMES" (VECTOR)
"STARTUP.LOSSES" (VECTOR)

Line 110: ENTITY 'RPM.SETTING:
"RPM.VALUE"

Line 111: SET 'TEST.POINTS'
ENTITY 'POINT:
"PCT.PEDAL"
"TORQUE"
"VACUUM.VALUE"
"FUEL.FLOW.RATE"
"HC.EMISSION.INDEX"
"CO.EMISSION.INDEX"
"NOX.VALUE"

Line 121: SET 'OTHER.LOSSES'
(SEE CHASSIS: OTHER.LOSSES DATA STRUCTURE)
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.


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
($) $5000
INFLATION RATE: .085

* 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

* SYSTEM *

TYPE: TELECOMMUNICATION & MESSAGE SWITCHING
REAL-TIME CODE: .15
LEVEL: 3
UTILIZATION: .60
SLIM SAMPLE OUTPUT

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

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

13.41 MONTHS
85 MANMONTHS
360 ($1000 \times $)

ENTER DESIRED DEV EFF IN MM?
50

NEW DEVELOPMENT TIME
SMALL TELECOM SYSTEM

<table>
<thead>
<tr>
<th></th>
<th>( \mu )</th>
<th>( \sigma )</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW DEV TIME</td>
<td>15.16</td>
<td>1.88</td>
</tr>
<tr>
<td>(MONTHS)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW DEV EFFORT</td>
<td>50</td>
<td>19</td>
</tr>
<tr>
<td>(MANMONTHS)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW DEV COST</td>
<td>208</td>
<td>86</td>
</tr>
<tr>
<td>($1000 \times $)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

CONSISTENCY CHECK
WITH INDEPENDENT DATA BASE

<table>
<thead>
<tr>
<th>Value</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 MM</td>
<td>IN NORMAL RANGE</td>
</tr>
<tr>
<td>15.2 MONTHS</td>
<td>IN NORMAL RANGE</td>
</tr>
<tr>
<td>3 AVG MNPWR</td>
<td>IN NORMAL RANGE</td>
</tr>
<tr>
<td>400 LINES/MM</td>
<td>IN NORMAL RANGE</td>
</tr>
</tbody>
</table>
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.

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


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.
ASSESSING THE PERFORMANCE OF THE CURRENT DESIGN

*** PRINCIPAL RESULTS ***

<table>
<thead>
<tr>
<th>WORKLOAD</th>
<th>RESPONSE TIME</th>
<th>THROUGHPUT</th>
<th>% CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 POLLING_WKL</td>
<td>0.01 SEC</td>
<td>36000. PER HOUR</td>
<td>7.5 %</td>
</tr>
<tr>
<td>2 STORE_WKL</td>
<td>1.59 SEC</td>
<td>6000. PER HOUR</td>
<td>25.0 %</td>
</tr>
<tr>
<td>3 FLASH_WKL</td>
<td>0.64 SEC</td>
<td>667. PER HOUR</td>
<td>4.2 %</td>
</tr>
<tr>
<td>4 FORWARD_WKL</td>
<td>1.25 SEC</td>
<td>1333. PER HOUR</td>
<td>6.9 %</td>
</tr>
</tbody>
</table>

TOTAL CPU UTILIZATION = 43.6%

SET FLASH_WKL ARRIVAL_RATE = 2000

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

*** PRINCIPAL RESULTS ***

<table>
<thead>
<tr>
<th>WORKLOAD</th>
<th>RESPONSE TIME</th>
<th>THROUGHPUT</th>
<th>% CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 POLLING_WKL</td>
<td>0.01 SEC</td>
<td>36000. PER HOUR</td>
<td>7.5 %</td>
</tr>
<tr>
<td>2 STORE_WKL</td>
<td>3.03 SEC</td>
<td>6000. PER HOUR</td>
<td>25.0 %</td>
</tr>
<tr>
<td>3 FLASH_WKL</td>
<td>0.83 SEC</td>
<td>2000. PER HOUR</td>
<td>12.5 %</td>
</tr>
<tr>
<td>4 FORWARD_WKL</td>
<td>1.52 SEC</td>
<td>1333. PER HOUR</td>
<td>6.9 %</td>
</tr>
</tbody>
</table>

TOTAL CPU UTILIZATION = 51.9%

SET FLASH_WKL ARRIVAL_RATE = 667

SET STORE_WKL ARRIVAL_RATE = 7500

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

*** PRINCIPAL RESULTS ***

<table>
<thead>
<tr>
<th>WORKLOAD</th>
<th>RESPONSE TIME</th>
<th>THROUGHPUT</th>
<th>% CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 POLLING_WKL</td>
<td>0.01 SEC</td>
<td>36000. PER HOUR</td>
<td>7.5 %</td>
</tr>
<tr>
<td>2 STORE_WKL</td>
<td>4.61 SEC</td>
<td>7500. PER HOUR</td>
<td>31.3 %</td>
</tr>
<tr>
<td>3 FLASH_WKL</td>
<td>0.66 SEC</td>
<td>667. PER HOUR</td>
<td>4.2 %</td>
</tr>
<tr>
<td>4 FORWARD_WKL</td>
<td>1.28 SEC</td>
<td>1333. PER HOUR</td>
<td>6.9 %</td>
</tr>
</tbody>
</table>

TOTAL CPU UTILIZATION = 49.9%
RESULTS OF THE PARAMETRIC STUDY PRESENTED GRAPHICALLY

**DGRAPH 1**

25-Nov-79 15:52
STORE_WKL RESPONSE TIME

**DGRAPH 2**

25-Nov-79 15:54
TOTAL CENTRAL_PROCESSOR UTIL
VARY STORE_WKL ARRIVAL_RATE

FROM 5000 TO 10000 BY 500

STOP IF 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

<table>
<thead>
<tr>
<th>STORE_WKL ARRIVAL_RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000.0</td>
</tr>
<tr>
<td>5500.0</td>
</tr>
<tr>
<td>6000.0</td>
</tr>
<tr>
<td>6500.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STORE_WKL RESPONSE_TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
</tr>
<tr>
<td>1.3</td>
</tr>
<tr>
<td>1.6</td>
</tr>
<tr>
<td>2.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TOTAL CENTRAL_PROCESSOR UTIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>39.4</td>
</tr>
<tr>
<td>41.5</td>
</tr>
<tr>
<td>43.6</td>
</tr>
<tr>
<td>45.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STORE_WKL ARRIVAL_RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>7000.0</td>
</tr>
<tr>
<td>7500.0</td>
</tr>
<tr>
<td>8000.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STORE_WKL RESPONSE_TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.9</td>
</tr>
<tr>
<td>4.6</td>
</tr>
<tr>
<td>12.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TOTAL CENTRAL_PROCESSOR UTIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>47.8</td>
</tr>
<tr>
<td>49.9</td>
</tr>
<tr>
<td>51.9</td>
</tr>
</tbody>
</table>
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., 470 TOTEN POND ROAD, WALTHAM, MA, 02254, USA, 617-890-0000
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 Lyklama, 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.nl.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 preparatory 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 headed 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 library 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:


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.

```plaintext
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
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, i, ii, 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(dedup(s(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 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 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 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 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 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 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 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: 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.

```
theoren inDedup, i in dedup(s) imp i in s;
```

proof tree:
```
  11: 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!)
```

```
theoren nodupsAfterDeduped, nodups(dedup(s));
nodupsAfterDeduped uses inDedup1.
```

proof tree:
```
  5: 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: 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:

```
SARA
  \_________\_________\_________\_________\_________\_________\_________
  |                   |                   |                   |                   |
  | requirements*     | behavior          | MID*              | library tutor     |
  |                   |                   |                   | utility comment   |
  |                   |                   |                   | news              |
  \_________\_________\_________\_________\_________\_________\_________
     |                   |                   |                   |
     |                   |                   |                   |
     | GMB                | SAM               |
     |                   |                   |
     \-------------------\-------------------
       translator       linker PLIP simulator analyzer
```

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 [PenR80]. 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-AF03-76SF0034 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:


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

c>u</n/SARA>SARA_system>c>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 gmb; 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 >u</n>SARA>sl>buffer
Working library now >u</n>SARA>sl>buffer

/* SABA STRUCTURE
SAB-LINKER Translator November 28, 1979

/* The source code for the buffer example
/* exists in the file buffer.all. The input
/* will be read in and echoed.
>input buffer.all -echo
Input from source >u</n>SARA>sl>buffer>buffer.all started

/* all definition for BUFFER example */
universe(environment, buffer_bb);
environment (<$write, $read>);
buffer_bb<$write, $read>;
universe($write : environment$write - buffer_bb$write);
universe($read : environment$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 SLL Translator

SARA

/* GMB DEFINITION

@behavior; gmb; @translator
SARA.Behavior.GMB
SARA.Behavior.GMB
GMB Translator Jan 16, 1980
/* Now we define gmb's (behavioral models)
/* to be associated with the structures.
>load all_model buffer /* load all_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(l), a1, a2, t,
acr, aw, af1,
acr, a3, a4, af2;
nt( af1*af2 : t);
```c
/* PLIP DEFINITION */
@plip buffer SARA.Behavior.GMB.PLIP
GMB PLI 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 PLI */
/* preprocessor; procedures are associated */
/* with data processors and types are */
/* associated with datasets/dataarcs. */
@system buffer_bb;

Current SLI 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(dpwin, dpwout, dprout)
tl fixed bin(15);
@dataset (ptr_w ptr_r) @like tl @initial(0);
@processor pw;

@read message @from din;
@read ptr_w @from dpwout;
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;

/* Start the simulation */
@start
messages were delivered correctly
End of simulation, time = 15200062 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
end of SARA Selector
```
FEATURE CLASSIFICATION:

INPUT
  . SUBJECT
    . VHL1 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 compatible 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:


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 Services 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

<table>
<thead>
<tr>
<th>CYBER/EKS TOOLS (ORIGIN)</th>
<th>IERAK TOOLS (ORIGIN)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Audit</strong> - PFORT (Bell Labs)</td>
<td>Management - Viewfoil Edit (SANA)</td>
</tr>
<tr>
<td><strong>Document</strong> - P003 (EKS/BACSD)</td>
<td>Milestone (SAMA), Forms Mode (SAMA)</td>
</tr>
<tr>
<td>CALLMAP (EKS/SAMA)</td>
<td>Design - Diagram Edit (SAMA)</td>
</tr>
<tr>
<td>DOCUMENTER (SAMA)</td>
<td>Data Dict (SAMA)</td>
</tr>
<tr>
<td><strong>Test</strong> - P003 (EKS/BACSD)</td>
<td>Programer</td>
</tr>
<tr>
<td>DYNAP (SAMA) FORTRAN</td>
<td>Toolbox - Customized Edit (SAMA)</td>
</tr>
<tr>
<td>DAPPER (BCAC/SAMA) PASCAL</td>
<td>PASCAL (ATAD)</td>
</tr>
<tr>
<td>COMMAP (SAMA)</td>
<td>Filer (ATAD)</td>
</tr>
<tr>
<td><strong>Design</strong> - PDL/FORTRAN (BCAC)</td>
<td>DAPPER - (BCAC/SAMA)</td>
</tr>
<tr>
<td></td>
<td>General</td>
</tr>
<tr>
<td></td>
<td>Utilities - Print (SANA)</td>
</tr>
<tr>
<td></td>
<td>Plot (SANA)</td>
</tr>
<tr>
<td></td>
<td>Display (SANA)</td>
</tr>
<tr>
<td><strong>LEGEND</strong></td>
<td></td>
</tr>
<tr>
<td><strong>CURRENT CAPABILITIES</strong></td>
<td></td>
</tr>
<tr>
<td><strong>FUTURE/PROPOSED CAPABILITIES</strong></td>
<td></td>
</tr>
</tbody>
</table>
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-03, SEATTLE, WA, 98124, USA, 206-575-5118
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 development 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.)


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 liaison 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.
### DYNA PROGRAM SUMMARY

**LAST RUN DATE:** 01/03/85  
**TIME:** 11.15.21  
**LIST DATE:** 01/03/85  
**TIME:** 11.17.00

---

#### SUMMARY OF 2 RUNS

**PROGRAM SOLVE**

---

#### CLASSIFICATION OF STATEMENTS

<table>
<thead>
<tr>
<th>STATEMENT TYPE</th>
<th>NUMBER PRESENT</th>
<th>PERCENT OF TOTAL</th>
<th>NUMBER MONITORED</th>
<th>PERCENT MONITORED</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMENT</td>
<td>12</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECLARATIVE</td>
<td>12</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXECUTABLE</td>
<td>37</td>
<td>61</td>
<td>37</td>
<td>100</td>
</tr>
<tr>
<td>UNRECOGNIZED</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>61</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

#### EXECUTABLE STATEMENTS

<table>
<thead>
<tr>
<th>STATEMENT TYPE</th>
<th>TOTAL STATEMENTS</th>
<th>EXECUTED</th>
<th>% EXECUTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSIGNMENT</td>
<td>11</td>
<td>9</td>
<td>82</td>
</tr>
<tr>
<td>DO</td>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>TRANSFER</td>
<td>19</td>
<td>11</td>
<td>65</td>
</tr>
<tr>
<td>SIMPLE GO TO</td>
<td>7</td>
<td>5</td>
<td>71</td>
</tr>
<tr>
<td>COMPUTED GO TO</td>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>BRANCHES</td>
<td>4</td>
<td>9</td>
<td>75</td>
</tr>
<tr>
<td>OUT-OF-RANGE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ASSIGN GO TO</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOGICAL IF</td>
<td>2</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>TRUE</td>
<td>2</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>FALSE</td>
<td>2</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>ARITHMETIC IF (3 BR)</td>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>NEGATIVE</td>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>ZERO</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>POSITIVE</td>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>ARITHMETIC IF (2 BR)</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZERO</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NON-ZERO</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL</td>
<td>2</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>CALL RETURNS</td>
<td>2</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>INPUT/OUTPUT</td>
<td>7</td>
<td>5</td>
<td>71</td>
</tr>
<tr>
<td>OTHER</td>
<td>5</td>
<td>4</td>
<td>80</td>
</tr>
</tbody>
</table>

---

#### SUMMARY TOTALS

<table>
<thead>
<tr>
<th>MODULES</th>
<th>EXECUTABLE</th>
<th>BRANCHES</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>37</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>81</td>
<td>75</td>
</tr>
<tr>
<td>MODULE NAME</td>
<td>TIMES ENTERED</td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td>PROGRAM SOLVE</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>SUBROUTINE LINEAR</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>SUBROUTINE DISCRM</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

3 OUT OF 3 MODULES ENTERED
100% MODULES ENTERED
### DYNA Module Summary

**Last Run Date:** 01/03/85  
**Time:** 11:15:21  
**List Date:** 01/03/85  
**Time:** 11:17:00

#### Summary of 2 Runs

**Subroutine Linear**

#### Classification of Statements

<table>
<thead>
<tr>
<th>Statement Type</th>
<th>Number Present</th>
<th>Percent of Total</th>
<th>Number Monitored</th>
<th>Percent Monitored</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Declarative</td>
<td>2</td>
<td>33</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>Executable</td>
<td>4</td>
<td>67</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>Unrecognized</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Total:** 6 Executable Statements

#### Statement Types

<table>
<thead>
<tr>
<th>Statement Type</th>
<th>Total Statements Present</th>
<th>Monitored Statements Executed</th>
<th>% Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment</td>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>Do</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Transfer</td>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>Simple Go To</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Computed Go To</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Branches</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Out-of-Range</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Assign Go To</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Logical If</td>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>True</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>False</td>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>Arithmetic If C3 Bro</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Negative</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Zero</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Positive</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Arithmetic If C2 Bro</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Zero</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Non-Zero</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Call</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Returns</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Input/Output</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Other</td>
<td>2</td>
<td>1</td>
<td>50</td>
</tr>
</tbody>
</table>

**Summary Totals**

- Executable Branches: 4 Executed: 3 % Executed: 75
- Branches: 2 Executed: 1 % Executed: 50

**Total Times Entered:** 2
**DYNA EXECUTION SUMMARY**

<table>
<thead>
<tr>
<th>Run Date</th>
<th>Run Time</th>
<th>Run Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/03/85</td>
<td>11.08.36</td>
<td>INPUT FILE: IN1</td>
</tr>
<tr>
<td>01/03/85</td>
<td>11.15.21</td>
<td>INPUT FILE: IN2</td>
</tr>
</tbody>
</table>

**DYNA ANNOTATED SOURCE LISTING**

<table>
<thead>
<tr>
<th>Line #</th>
<th>Source</th>
<th>Monitor Counts</th>
<th>Total</th>
<th>Conditional</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PROGRAM SOLVE(INFIL. OUTFIL. T...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>INTEGER COUNT, JUMP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>081 FORMAT( I10 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>082 FORMAT(2F10.2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>061 FORMAT(1X,F10.2, &quot; X==2 +&quot;, F1...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>* /,5X,&quot;X1 = &quot;,F10.2, &quot;...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>* /,5X,&quot;X2 = &quot;,F10.2, &quot;...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>052 FORMAT(1X,F10.2, &quot; X==2 +&quot;, F1...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>* /,5X,&quot;X1 = &quot;,F10.2,/...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>063 FORMAT(1X,F10.2, &quot; = 0&quot;,/5X,...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>054 FORMAT(1X,F10.2, &quot; = 0&quot;,/5X,...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>READ(7, 081) COUNT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>DO 100 I=1, COUNT</td>
<td></td>
<td></td>
<td>EXECUTED: 6</td>
</tr>
<tr>
<td>15</td>
<td>READ(7, 082) A, B, C</td>
<td></td>
<td></td>
<td>TRUE: 3</td>
</tr>
<tr>
<td>16</td>
<td>IF (A .NE. 0.) GOTO 10</td>
<td></td>
<td></td>
<td>FALSE: 3</td>
</tr>
<tr>
<td>17</td>
<td>CALL LINEAR(B, C, X), RET...</td>
<td></td>
<td>RETURNED: 2</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>JUMP = 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>GOTO 50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>10 CALL DISCRM(A, B, C, D)</td>
<td></td>
<td></td>
<td>RETURNED: 3</td>
</tr>
<tr>
<td>22</td>
<td>IF (D 20, 30, 40)</td>
<td></td>
<td></td>
<td>NEGATIVE: 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ZERO: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>POSITIVE: 1</td>
</tr>
<tr>
<td>23C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>20 REALX = -B/(2*A)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>COMPLEX = SQRT(-D)/(2*A)...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>JUMP = 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>GOTO 50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>30 REALX = -B/(2*A)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>JUMP = 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>GOTO 50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINE#</td>
<td>SOURCE</td>
<td>TOTAL</td>
<td>CONDITIONAL</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>-------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>REAL1 = (-B + SQRT(D))...</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>REAL2 = (-B - SQRT(D))...</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>JUMP = 3</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36C</td>
<td>GOTO (60, 70, 80, 90), JUMP</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>WRITE (9, 951) A, B, C, REALS...</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>GOTO 100</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41C</td>
<td>WRITE (9, 952) A, B, C, REALX, R...</td>
<td><strong>@</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>GOTO 100</td>
<td><strong>@</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44C</td>
<td>WRITE (9, 952) A, B, C, REAL1,...</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>GOTO 100</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>47C</td>
<td>WRITE (9, 953) B, C, X</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>GOTO 100</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50C</td>
<td>WRITE (9, 954) C</td>
<td><strong>@</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52C</td>
<td>100 CONTINUE</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>STOP</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>END</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>SUBROUTINE LINEAR B, C, X, REAL...</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>REAL B, C, X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>IF (B .EQ. 0) GO TO 10</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>X = -C/B</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>RETURN M</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>61 10</td>
<td>RETURN N</td>
<td><strong>@</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>END</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>SUBROUTINE DISCRM (A, B, C, D)</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>REAL A, B, C, D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>D = B<em>B - 4</em>A*C</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>RETURN</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>END</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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-03, SEATTLE, WA, 98124, USA, 206-575-5118
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.)

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

**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

<table>
<thead>
<tr>
<th>Variable</th>
<th>STRIP 1</th>
<th>KEY</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEV</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GP I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PV</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FIREW</strong>K</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>HEIGH</strong>T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>XANGLE</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>XSPEED</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>YANGLE</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>YSPEED</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PHYCON</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>QUAD</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RCOMP</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XDIST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>XGVEL</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YDIST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YGUEL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SPACEN</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ELYSP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LKUNI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVEL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- **++** indicates a warning for equivalenced variables.
- **+++** indicates a common block defined value with multiple definitions.
## COMMAP VARIABLE WARNING SUMMARY

**PAGE 1**

<table>
<thead>
<tr>
<th>VARIABLES</th>
<th>VARIABLES</th>
<th>VARIABLES</th>
<th>VARIABLES</th>
<th>VARIABLES</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFERENCED</td>
<td>DEFINED</td>
<td>WITH</td>
<td>NEVER</td>
<td>USED</td>
<td>ONLY ONE</td>
</tr>
<tr>
<td>COMMON</td>
<td>BUT NEVER</td>
<td>BUT NEVER</td>
<td>WARNINGS*</td>
<td>USED</td>
<td>MODULE*</td>
</tr>
<tr>
<td>BLOCKS</td>
<td>DEFINED</td>
<td>REFERENCED</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**List Date:** 01/03/05  
**Time:** 09:20:26

### **FIREWK**
- **HEIGHT**
- **XANGLE**
- **XSPEED**
- **YANGLE**
- **YSPEED**

### **PHYCON**
- **PI**
- **G**
- **PI**

### **QUAD**
- **C**
- **B**
- **A**
- **A**

### **RCOMP**
- **XGVEL**
- **XDIST**
- **YDIST**

### **SPACEN**
- **MVEL**
- **ELVSP**
- **MVEL**

**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: 01/03/05.

PAGE 1

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: 01/03/05.

NECESSARY COMMON BLOCK DECLARATIONS

COMMON :
BLOCKS : MODULES USING VARIABLES

PHYCON : OUTPUT
QUAD : SETUP DISC
RCOMP : SETUP INPUT DISC LINEAR VEL :
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,

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 Flowchart 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 Flowchart. 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
         ENDFOR
         ENDFUTHIF
      ENDFELSE
   EXITPROC
ENDPROC
PROCEDURE DL_EXAMPLE

ENTER

CONDO

TEXT5 COND1 I=1

VAR=1,2

TEXT1 COND1 TEXT2

VAR=3,7 TEXT4

COND2 TEXT3

EXIT

TEXT6 I=I+1 I≤7

I=1
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

ENTER

\[ \text{OBJNUM} = \text{OBJNUM} + 1 \]
\[ \text{OBJINT} = \text{OBJINT} + 1 \]

10
\[ \text{TOBJ} = \text{OBJTYP}(\text{OBJNUM} + \text{TRNUM} - 1) \]

T3
\[ \text{OBJ} = 'EOT' \]

T4
\[ \text{OBJ} = 'RV' \]

T5
\[ \text{OBJ} = 'RV/CH' \]

T6
\[ \text{OBJ} = 'CH' \]

20
\[ \text{TRNUM} = \text{TRNUM} + 1 \]
\[ \text{OBJINT} = 1 \]

30
\[ \text{RVNUM} = \text{RVNUM} + 1 \]

40
\[ \text{RVNUM} = \text{RVNUM} + 1 \]
\[ \text{CHNUM} = \text{CHNUM} + 1 \]

50
\[ \text{CHNUM} = \text{CHNUM} + 1 \]

S7

60
\[ \text{I} = 0 \]
\[ \text{NOBJTR} = \text{TRNUM} \]

S11

70
\[ \text{NOBJ} = \text{OBJTYP}(\text{OBJNUM} + \text{TRNUM} + 1) \]

T13
\[ \text{NOBJ} = 'EOT' \]

S14
\[ \text{NOBJTR} = \text{NOBJTR} + 1 \]
\[ \text{I} = \text{I} + 1 \]
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
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


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 *).
OXX 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
(*400x*).
STRUCTURE:
    INPUT_INTERFACE: FROM_OPERATOR
    CONSIDER DATA: OPERATOR_MSG_TYPE
    IF (REESTABLISH_REGISTRAR_TERMINAL)
        ALPHA: TURN_OFF_INP_INPUT_PROHIBITED_FLAG
        TERMINATE
    OR (SET_SEMESTER)
        ALPHA: SET_NEW_SEMESTER
        TERMINATE
    OR (SET_YEAR)
        ALPHA: SET_NEW_YEAR
        TERMINATE
    END
END.
CRADX Command:

`ANALYZE DATA_FLOW COMPUTER_OPERATOR_PROCESSING.`
`ANALYZE DATA_FLOW 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 YEAR_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


"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.
LINE DRIVER

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

Abstract Data Types in SDP

1. DECLARE QUEUE_LENGTH AS COUNTER
2. DECLARE BUFFER_CONTROL_BLOCK
3. DECLARE HEAD AS A PCINTER
4. DECLARE TAIL AS A PCINTER
5. OP QUEUE NOT EMPTY
   - RETURNS TRUE OR FALSE
   - IF QUEUE_LENGTH = 0
   - THEN
   - RETURN FALSE
   - ELSE
   - RETURN TRUE
   - FI
7. OP GET NEXT BUFFER_CONTROL_BLOCK
   - IF QUEUE_LENGTH < 0
   - THEN
   - REMOVE HEAD BUFFER_CONTROL_BLOCK
   - DECREMENT QUEUE_LENGTH
   - ELSE
   - RETURN BUFFER_CONTROL_BLOCK
   - ELSE
   - ERROR
   - FI
9. OP APPEND BUFFER_CONTROL_BLOCK TO QUEUE
10. INSERT BUFFER_CONTROL_BLOCK AT TAIL OF QUEUE
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, 054-58534
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 chemistry 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
ENTER E
SUB1 M
          
COM2 L
COM1 C
INSERT U
SUB3 U
SUB4 M
          
SUB3 U
SUB4 M
          
SUB5 M
          **RECURSION**
          
SUB3 U
SUB6 M
          
SEARCH E
ENTER E
COM2 C
INSERT U
SUB3 M
SUB2 M
          **REPEATED NODE**
```

Sample Test Coverage Report - Routine level

**************************************************************************
* TEST COVERAGE DOCUMENTATION *
**************************************************************************

TEST COVERAGE PROFILE

TOTAL ROUTINES IN SYSTEM: 14

<table>
<thead>
<tr>
<th>NUMBER OF ROUTINES EXERCISED</th>
<th>% OF TOTAL ROUTINES</th>
<th>NUMBER OF ROUTINE EXECUTIONS</th>
<th>TEST EFFECTIVENESS (NUMBER EXECUTIONS/ % TOTAL ROUTINES)</th>
</tr>
</thead>
<tbody>
<tr>
<td>THIS RUN:</td>
<td>4</td>
<td>28.57</td>
<td>46</td>
</tr>
</tbody>
</table>

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

**OPTIMIZATION DOCUMENTATION**

<table>
<thead>
<tr>
<th>STATEMENT NUMBER</th>
<th>TOTAL TIME (mSEC)</th>
<th>% OF TOTAL TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>56.320</td>
<td>35.53</td>
</tr>
<tr>
<td>11</td>
<td>38.408</td>
<td>24.23</td>
</tr>
<tr>
<td>2</td>
<td>28.931</td>
<td>18.25</td>
</tr>
<tr>
<td>9</td>
<td>13.045</td>
<td>8.23</td>
</tr>
<tr>
<td>6</td>
<td>10.595</td>
<td>6.68</td>
</tr>
<tr>
<td>41</td>
<td>5.028</td>
<td>3.17</td>
</tr>
<tr>
<td>4</td>
<td>1.550</td>
<td>.98</td>
</tr>
<tr>
<td>27</td>
<td>.896</td>
<td>.57</td>
</tr>
<tr>
<td>24</td>
<td>.576</td>
<td>.36</td>
</tr>
<tr>
<td>33</td>
<td>.504</td>
<td>.32</td>
</tr>
<tr>
<td>12</td>
<td>.496</td>
<td>.31</td>
</tr>
<tr>
<td>7</td>
<td>.421</td>
<td>.27</td>
</tr>
<tr>
<td>22</td>
<td>.288</td>
<td>.18</td>
</tr>
</tbody>
</table>

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 - 270K BYTES, TOOL AVAILABLE: YES, PUBLIC
DOMAIN: NO, RESTRICTIONS (COPYRIGHTS, LICENSES, ETC.):
PROPRIETARY PRODUCT, TOOL SUPPORTED: YES, TOOL
SUPPORT: SOFTOOL CORPORATION

CONTACT: SOFTOOL CORPORATION, 340 S. KELLOGG, GOLETA, CA,
93117, USA, 805-964-0560
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.


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
6 ISUM = 0
7 IF (N .GT. 1)
8 ( 1) IF
9 ( 1) I = 1
10 ( 2) WHILE (I .LE. N)
11 ( 2) ISUM = ISUM + A(I)
12 ( 1) I = I + 1
13 ( 1) END WHILE
14 ELSE
15 N
16 END IF
17 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. Cl: 83.33%
(Previous tests) 0 segments hit out of 6. Cl: .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. Cl: 50.00%
(Previous tests) 5 segments hit out of 6. Cl: 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. Cl: 50.00%
(Previous tests) 6 segments hit out of 6. Cl: 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 Cl 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

- 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.


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, Maryland 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 from 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  MINCH=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 I0PAR/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,ITKLIST,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 1
UNSUBSCRIPTED OCCURRENCE OF VARIABLE IN STATEMENT 20, NO CHANGE MADE.
      240 DIMENSION ITKDIR(200)
      310 ITKDIR(ITKLIST)=ISTART
      320 ITKDIR(ITKLIST)=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 IGRP=IGRP(K)
1440 INDTY=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 IGRP=IGRP(K+1)
1440 INDTY=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
830 320 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 ILST=I
320 ITBLPR=1
330 300 CONTINUE

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.): LICENSED 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.


- 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.
Sample Output

<table>
<thead>
<tr>
<th>Keyname</th>
<th>Longname</th>
<th>Datatype</th>
<th>Part of</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>INFORMATION FILE</td>
<td>ORDSET</td>
<td></td>
</tr>
<tr>
<td>SSF</td>
<td>SEARCH SCAN FILE</td>
<td>ORDSET</td>
<td></td>
</tr>
<tr>
<td>RAF</td>
<td>RADAR ACTIVITIES FILE</td>
<td>ORDSET</td>
<td></td>
</tr>
<tr>
<td>HM</td>
<td>HANDOVER MESSAGE</td>
<td>ORDSET</td>
<td></td>
</tr>
<tr>
<td>SQ</td>
<td>ORDERED SET OF REQUESTS</td>
<td>ORDSET</td>
<td></td>
</tr>
</tbody>
</table>

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


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-graphics 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

\[ S_1 \quad B \quad \ldots \ldots \quad S_1 \quad S_2 \quad * \quad S_1 \quad B \quad \triangleleft \text{in} \quad \triangleright \text{out} \]

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.
SCHEMACODE

1. A. b, c

2. COEFFICIENTS ARE. a, b, c

3. a = o

4. b = o

5. 'NO ROOT'

6. -c/b

7. d = -b^2 - 4ac

8. d > o

9. -b + \sqrt{d} / 2a

10. d = o

11. -b / 2a

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 remaining 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.
SOLUTION OF QUADRATIC EQUATION
GET ONE SET OF COEFFICIENTS
\( [A, B, C] \)

EOF

"COEFFICIENTS", A, B, C

COMPUTE ROOTS
R001
\( [A, B, C] \)

R000

compute roots

A = 0

linear
R002

quadratic
R003

R001
SCHEMACODE
SAMPLE OUTPUT

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

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

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
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 circumstances. 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 examined 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. Usually 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

[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 Science
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
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:

```
  SARA
  |    |
requirements* SII behavior MID* |
  |    |
library tutor utility comment news
  |    |
GMB SAM
  |    |
translator linker PLIP simulator analyzer
```

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-AF03-76SF0034 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:


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 Assistant 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 Engineering 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  Lynne A. Price
SPERRY·UNIVAC  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.
<table>
<thead>
<tr>
<th>USER MAY ENTER</th>
<th>TO OBTAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>?ERROR</td>
<td>successively more detailed explanations of a displayed error message</td>
</tr>
<tr>
<td>?QUESTION</td>
<td>successively more detailed explanations of a displayed question or prompt</td>
</tr>
<tr>
<td>?EXAMPLE</td>
<td>successive examples of correct input or valid commands</td>
</tr>
<tr>
<td>?DEFINE term</td>
<td>explanation or definition of a specified term</td>
</tr>
<tr>
<td>?FORMAT command</td>
<td>a description of the format of a specified command</td>
</tr>
<tr>
<td>?MENU</td>
<td>a list of allowable commands</td>
</tr>
<tr>
<td>?DOC section</td>
<td>a display of a specified section of documentation</td>
</tr>
<tr>
<td>?STATUS</td>
<td>a description of the current value of various system parameters</td>
</tr>
<tr>
<td>?INSTRUCT</td>
<td>instruction on the use of the system</td>
</tr>
<tr>
<td>?NEWS</td>
<td>news of interest to users of the system</td>
</tr>
<tr>
<td>?HELP</td>
<td>a list of available user aids</td>
</tr>
</tbody>
</table>

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.

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 Department, 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?  [A]
Enter one of the following for help:
   ?error, ?menu, ?request KEYWORD, ?define TERM
For explanation of these help functions enter
   ?
again.
Request?  [P]
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 (or ?r KEYWORD)  a description of the indicated request (KEYWORD
                is one of the request keywords listed by ?m)

?define TERM (or ?d TERM)  a definition of the indicated term (if TERM is
                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.
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:
- key call transfer
- call transfer lamp
- 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
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 statement 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


"FORTRAN 77 Analyzer", Maintenance Manual, TRW Defense and Space Systems Group, One Space Park, Redondo Beach, CA 90278

5. Place and Time

<table>
<thead>
<tr>
<th>Station</th>
<th>Day</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Thursday, March 12</td>
<td>9 AM to 3 PM</td>
</tr>
</tbody>
</table>

6. Demonstrator

Patricia Powell
National Bureau of Standards
BLDG 225 RM A265
Washington, D.C. 20234
(301) 921-3435

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

<table>
<thead>
<tr>
<th>MODULE NAME</th>
<th>NUMBER OF STATEMENTS</th>
<th>PERCENT STATEMENT COVERAGE</th>
<th>NUMBER OF STATEMENTS COMMENTS</th>
<th>NUMBER OF SYNTAX ERRORS</th>
<th>INVOCATION FREQUENCY</th>
<th>NUMBER OF SEGMENTS</th>
<th>PERCENT SEGMENT COVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td>119</td>
<td>0.95</td>
<td>66</td>
<td>0</td>
<td>2</td>
<td>53</td>
<td>0.96</td>
</tr>
<tr>
<td>SHUFFL</td>
<td>17</td>
<td>1.00</td>
<td>8</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1.00</td>
</tr>
<tr>
<td>DEAL</td>
<td>12</td>
<td>1.00</td>
<td>5</td>
<td>0</td>
<td>26</td>
<td>3</td>
<td>1.00</td>
</tr>
<tr>
<td>PLAY</td>
<td>25</td>
<td>1.00</td>
<td>10</td>
<td>0</td>
<td>94</td>
<td>7</td>
<td>0.86</td>
</tr>
<tr>
<td>EVALLC</td>
<td>20</td>
<td>0.00</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0.00</td>
</tr>
<tr>
<td>MATCH</td>
<td>20</td>
<td>1.00</td>
<td>0</td>
<td>0</td>
<td>94</td>
<td>8</td>
<td>1.00</td>
</tr>
<tr>
<td>LARGE</td>
<td>12</td>
<td>0.00</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0.00</td>
</tr>
<tr>
<td>RMHAN</td>
<td>17</td>
<td>1.00</td>
<td>5</td>
<td>0</td>
<td>96</td>
<td>5</td>
<td>1.00</td>
</tr>
<tr>
<td>RMCFBD</td>
<td>17</td>
<td>1.00</td>
<td>5</td>
<td>0</td>
<td>52</td>
<td>4</td>
<td>1.00</td>
</tr>
<tr>
<td>BUILD</td>
<td>43</td>
<td>0.97</td>
<td>9</td>
<td>0</td>
<td>50</td>
<td>14</td>
<td>0.93</td>
</tr>
<tr>
<td>DISCAR</td>
<td>26</td>
<td>1.00</td>
<td>11</td>
<td>0</td>
<td>46</td>
<td>9</td>
<td>1.00</td>
</tr>
</tbody>
</table>

### SUMMARY TOTALS

<table>
<thead>
<tr>
<th>MODULE NAME</th>
<th>NUMBER OF STATEMENTS</th>
<th>PERCENT STATEMENT COVERAGE</th>
<th>NUMBER OF STATEMENTS COMMENTS</th>
<th>NUMBER OF SYNTAX ERRORS</th>
<th>INVOCATION FREQUENCY</th>
<th>NUMBER OF SEGMENTS</th>
<th>PERCENT SEGMENT COVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRM</td>
<td>328</td>
<td>0.89</td>
<td>137</td>
<td>0</td>
<td>462</td>
<td>114</td>
<td>0.89</td>
</tr>
</tbody>
</table>

### PROGRAM CALL TREE

```
1  MAIN
  2  SHUFFL
  3  RAN
  4  DEAL
  5  DISCAR
  6  RMHAN
  7  RMCFBD
  8  PLAY
  9  DISCAR (5)
 10  RMCFBD
 11  MATCH
 12  BUILD
 13  DISCAR (5)
 14  RMHAN
 15  RMHAN
 16  EVALC
 17  LARGE
```
## SEGMENT EXECUTION FREQUENCIES - CURRENT

<table>
<thead>
<tr>
<th>Segment</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td></td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>52</td>
</tr>
<tr>
<td>0X</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>1X</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>16</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>2X</td>
<td>17</td>
<td>68</td>
<td>68</td>
<td>3</td>
<td>14</td>
<td>56</td>
<td>6</td>
<td>50</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3X</td>
<td>4</td>
<td>3</td>
<td>10</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>4X</td>
<td>5</td>
<td>8</td>
<td>8</td>
<td>47</td>
<td>49</td>
<td>47</td>
<td>49</td>
<td>56</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>5X</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHUFFL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>51</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>5X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13</td>
<td>52</td>
<td>52</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLAY</td>
<td></td>
<td>47</td>
<td>47</td>
<td>188</td>
<td>25</td>
<td>22</td>
<td>47</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6X</td>
<td></td>
<td>47</td>
<td>47</td>
<td>188</td>
<td>25</td>
<td>22</td>
<td>47</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EVALC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATCH</td>
<td></td>
<td>47</td>
<td>117</td>
<td>340</td>
<td>28</td>
<td>24</td>
<td>4</td>
<td>316</td>
<td>316</td>
<td></td>
</tr>
<tr>
<td>7X</td>
<td></td>
<td>47</td>
<td>117</td>
<td>340</td>
<td>28</td>
<td>24</td>
<td>4</td>
<td>316</td>
<td>316</td>
<td>0</td>
</tr>
<tr>
<td>LARGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMHAN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMCFBD</td>
<td></td>
<td>48</td>
<td>21</td>
<td>27</td>
<td>41</td>
<td>41</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8X</td>
<td></td>
<td>48</td>
<td>21</td>
<td>27</td>
<td>41</td>
<td>41</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BUILD</td>
<td></td>
<td>26</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9X</td>
<td></td>
<td>26</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9X</td>
<td></td>
<td>26</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BUILD</td>
<td></td>
<td>26</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISCAR</td>
<td></td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>10X</td>
<td></td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>DISCAR</td>
<td></td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>10X</td>
<td></td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>10X</td>
<td></td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>11X</td>
<td></td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>11X</td>
<td></td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
</tbody>
</table>

## SEGMENTS NOT EXECUTED

<table>
<thead>
<tr>
<th>Segments</th>
<th>37</th>
<th>51</th>
<th>61</th>
<th>67</th>
<th>68</th>
<th>69</th>
<th>70</th>
<th>79</th>
<th>80</th>
<th>81</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>51</td>
<td>61</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>79</td>
<td>80</td>
<td>81</td>
<td>99</td>
</tr>
</tbody>
</table>
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
    .  2. 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.
<table>
<thead>
<tr>
<th>WHAT PROCEDURE</th>
<th>ENTER</th>
<th>PRINT</th>
<th>RETRIEVE</th>
<th>SORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETRIEVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENTER FEATURE DESCRIPTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;COVERAGE ANALYSIS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30 TOOLS FOUND</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE TAXONOMY, YES OR NO? NO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHAT PROCEDURE :</td>
<td>ENTER</td>
<td>PRINT</td>
<td>RETRIEVE</td>
<td>SORT</td>
</tr>
<tr>
<td>RETRIEVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENTER FEATURE DESCRIPTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;ASSERTION CHECKING</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 TOOLS FOUND</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE TAXONOMY, YES OR NO? NO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHAT PROCEDURE :</td>
<td>ENTER</td>
<td>PRINT</td>
<td>RETRIEVE</td>
<td>SORT</td>
</tr>
<tr>
<td>RETRIEVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENTER FEATURE DESCRIPTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;ASSERTION CHECKING &amp; COVERAGE ANALYSIS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32 TOOLS FOUND</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE TAXONOMY, YES OR NO? NO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHAT PROCEDURE :</td>
<td>ENTER</td>
<td>PRINT</td>
<td>RETRIEVE</td>
<td>SORT</td>
</tr>
<tr>
<td>RETRIEVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENTER FEATURE DESCRIPTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;ASSERTION CHECKING &amp; COVERAGE ANALYSIS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 TOOLS FOUND</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE TAXONOMY, YES OR NO? NO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHAT PROCEDURE :</td>
<td>ENTER</td>
<td>PRINT</td>
<td>RETRIEVE</td>
<td>SORT</td>
</tr>
<tr>
<td>NOTHING</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NBS SOFTWARE TOOLS DATABASE

WHAT PROCEDURE: ENTER PRINT RETRIEVE SORT

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

WHAT SHOULD BE TYPED: ABSTRACT BIBLIOGRAPHY CONTACT DOCUMENTATION
ENVIRONMENT HARDWARE IDENTITY LANGUAGE
MAKER PROGRESS REFERENCES SOFTWARE
TAXONOMY USEABILITY OR NOTHING?

IDENTITY
ACRONYM TITLE

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
<table>
<thead>
<tr>
<th>WHAT SHOULD BE TYPED</th>
<th>ABSTRACT</th>
<th>BIBLIOGRAPHY</th>
<th>CONTACT</th>
<th>DOCUMENTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ENVIRONMENT</td>
<td>HARDWARE</td>
<td>IDENTITY</td>
<td>LANGUAGE</td>
</tr>
<tr>
<td></td>
<td>MAKER</td>
<td>PROGRESS</td>
<td>REFERENCES</td>
<td>SOFTWARE</td>
</tr>
<tr>
<td></td>
<td>TAXONOMY</td>
<td>USEABILITY OR NOTHING?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MAKER**

**ACRONYM**

**DEVELOPER**

---

**FTN-77 ANALYZER**

**TRW, INC.**

**TOOLS DATABASE**

**NATIONAL BUREAU OF STANDARDS**

<table>
<thead>
<tr>
<th>WHAT SHOULD BE TYPED</th>
<th>ABSTRACT</th>
<th>BIBLIOGRAPHY</th>
<th>CONTACT</th>
<th>DOCUMENTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ENVIRONMENT</td>
<td>HARDWARE</td>
<td>IDENTITY</td>
<td>LANGUAGE</td>
</tr>
<tr>
<td></td>
<td>MAKER</td>
<td>PROGRESS</td>
<td>REFERENCES</td>
<td>SOFTWARE</td>
</tr>
<tr>
<td></td>
<td>TAXONOMY</td>
<td>USEABILITY OR NOTHING?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LANGUAGE**

**ACRONYM**

**LANGUAGE**

**DIALECT**

---

**FTN-77 ANALYZER**

**FORTRAN**

**FORTRAN 77**

**TOOLS DATABASE**

**PASCAL**

R

<table>
<thead>
<tr>
<th>WHAT SHOULD BE TYPED</th>
<th>ABSTRACT</th>
<th>BIBLIOGRAPHY</th>
<th>CONTACT</th>
<th>DOCUMENTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ENVIRONMENT</td>
<td>HARDWARE</td>
<td>IDENTITY</td>
<td>LANGUAGE</td>
</tr>
<tr>
<td></td>
<td>MAKER</td>
<td>PROGRESS</td>
<td>REFERENCES</td>
<td>SOFTWARE</td>
</tr>
<tr>
<td></td>
<td>TAXONOMY</td>
<td>USEABILITY OR NOTHING?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTHING**

**WHAT PROCEDURE**

**ENTER**

**PRINT**

**RETRIEVE**

**SORT**

**TYPE OR 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


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

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 Hiltersvei 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 (WOCZKOW 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 HOLTHERSVEI 2, KOLBOTN, N-1410, NORWAY, 47 2-80 16 97
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 Installation Guide" - A guide to bringing up, "bootstrapping", the
UCSD p-System.


In addition, the following papers will be of interest:


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 enhancing a P-code interpreter and p-System facilities for Dr. Kenneth L. Bowles at the University of California, San Diego.

UCSD, UCSD Pascal, and UCSD p-System are all trademarks of the Regents of the University of California.
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


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
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.
PROGRAM KEYST

1. PROGRAM KEYST
2. C.KEYWORD=ON
3. LOOP FOREVER
4. FOR FIRST/NEXT CARD
5. EXIT IF NO MORE CARDS
6. FOR CARD COLUMNS 1 THROUGH 10
7. IF CURRENT CARD COLUMN CONTAINS A SPECIAL CHARACTER
8. CALL ROUTINE TO ENCRYPT THE CHARACTER
9. END IF
10. END FOR
11. SAVE ENCRYPTED CARD FOR LATER OUTPUT
12. END FOR
13. END LOOP
14. IF ANY CARDS WERE ENCRYPTED
15. OUTPUT THE CARDS TO THE SAVE FILE FOR LATER PROCESSING
16. STOP
17. END

FIGURE 1

IFTRAN™

PROGRAM DESIGN LANGUAGE Example

C.SAVE NAME
C. THIS IS MODULE ######.
C.END
C.SAVE TYPE COMMON /TYPE/ KIND
C.END

PROGRAM KEYST

DD PATH 1 IS ENTER DECK
C-END

FIGURE 2(a)

IFTRAN™

Example Usage of the BLOCK COPYING, MACRO, INSTRUMENTATION,
TESTBOUND, STATEMENT COMMENTING, and INDENTED LISTING
Features
(MAIN PROGRAM)
FUNCTION SQROOT (ARG)

  C.LIST=OFF
  C.Macro Name/##/SQROOT/
  C. THIS IS MODULE SQROOT.
  C.END

  C.END
  C.Macro Type
  C.COMMON /TYPE/ KIND

  C.END

  C.ENTRY=SQROOT

  BARG=ARG
  INITIAL (ARG.GT.0.0), FAIL (ERROR)
  SQROOT=SQRT(BARG)
  RETURN

  DD PATH 1 IS ENTER DECK

  DD PATH 2 IS RETURN

  DD PATH 3 IS ENTER BLOCK

  DD PATH 4 IS EXIT BLOCK

END

******************************************************************************

CROSS-REFERENCE

---

<table>
<thead>
<tr>
<th>BLOCK NAME</th>
<th>DEFINED</th>
<th>INVOKED</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR</td>
<td>16</td>
<td>15</td>
</tr>
</tbody>
</table>
---

FIGURE 2(b)

Example Usage of the
BLOCK COPYING, MACRO, INSTRUMENTATION,
EXECUTABLE ASSERTIONS, and INDEDITED LISTINGS
Features

(SUBPROGRAM)

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
AND FUNCTION SQROOT VALUE RETURNED EQUAL TO ZERO.
THE SQUARE ROOT OF -34.00000 IS 0.000000E+00
ENTER THE NUMBER TO BE SQUARE ROOTED.

1200 THE SQUARE ROOT OF 1200.000 IS 34.64102
END EXECUTION

FIGURE 2(c)

Sample Output from the
Execution of the Programs in Figures 2(a) & 2(b)

Illustrating the EXECUTABLE ASSERTIONS Feature
### Sample DETAILED INSTRUMENTATION REPORT

**For Program in Figure 2(a)**

<table>
<thead>
<tr>
<th>DD PATH NO.</th>
<th>NO. NOT EXECUTED</th>
<th>NUMBER OF EXECTIONS -- NORMALIZED TO MAXIMUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX</td>
</tr>
</tbody>
</table>

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(b)**

<table>
<thead>
<tr>
<th>DD PATH NO.</th>
<th>NO. NOT EXECUTED</th>
<th>NUMBER OF EXECTIONS -- NORMALIZED TO MAXIMUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX</td>
</tr>
</tbody>
</table>

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(c)**

<table>
<thead>
<tr>
<th>CASE NAME</th>
<th>NUMBER OF D-D PATHS</th>
<th>INVOCATIONS</th>
<th>TRAVERSED</th>
<th>PER CENT COVERAGE</th>
<th>NUMBER OF TESTS</th>
<th>INVOCATIONS</th>
<th>TRAVERSED</th>
<th>COVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IFTEST</td>
<td>8</td>
<td>2</td>
<td>37.50</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>50.00</td>
</tr>
<tr>
<td></td>
<td>SOROOT</td>
<td>4</td>
<td>1</td>
<td>50.00</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>50.00</td>
</tr>
<tr>
<td></td>
<td>$ALL##</td>
<td>12</td>
<td>5</td>
<td>41.67</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>41.67</td>
</tr>
</tbody>
</table>

**FIGURE 3(c)**

**IFTRAN**

Sample SUMMARY INSTRUMENTATION REPORT

For Programs in Figures 2(a) & 2(b)

(Multiple Test Cases due to TESTBOUND in Figure 2(a))

<table>
<thead>
<tr>
<th>MODULE</th>
<th>TEST</th>
<th>PATHS</th>
<th>NAME</th>
<th>NUMBER</th>
<th>NOT HIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFTEST</td>
<td>CUMUL</td>
<td>8</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>SOROOT</td>
<td>CUMUL</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

**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)

<table>
<thead>
<tr>
<th>STMT</th>
<th>NEST</th>
<th>LINE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>SUBROUTINE CIRCLE (AREA)</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>INTEGER AREA</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>RADIUS=DIAMTR/2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>AREA=PI*RADIUS/2</td>
</tr>
</tbody>
</table>

- VARIABLE DIAMTR: USED BUT NEVER SET
- SET/USE ERROR
- REFER TO STATEMENT(S):

<table>
<thead>
<tr>
<th>STMT</th>
<th>NEST</th>
<th>LINE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>IF (AREA&lt;0.5)</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>CALL PRINT (AREA)</td>
</tr>
</tbody>
</table>

- MODE WARNING
- PARAMETER 1 OF PRINT: ACTUAL PARAMETER HAS MODE INTEGER
- CALL ERROR: PRINT CALLED WITH 1 ACTUALLY HAS 2 ARGUMENTS

<table>
<thead>
<tr>
<th>STMT</th>
<th>NEST</th>
<th>LINE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>END IF</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

- STATEMENT 10 IS UNREACHABLE OR IS IN AN INFINITE LOOP

#### Static Analysis Summary

<table>
<thead>
<tr>
<th>Errors</th>
<th>Warnings</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

CALL CHECKING WAS NOT PERFORMED FOR THE FOLLOWING UNKNOWN EXTERNALS...

### Static Analysis

#### Subroutine PRINT (RADIUS, AREA)

<table>
<thead>
<tr>
<th>STMT</th>
<th>NEST</th>
<th>LINE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>SUBROUTINE PRINT (RADIUS, AREA)</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>WRITE (*,100) RADIUS, AREA</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STMT</th>
<th>NEST</th>
<th>LINE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>PRINT 3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>PRINT 5</td>
</tr>
</tbody>
</table>

#### Static Analysis Summary

<table>
<thead>
<tr>
<th>Errors</th>
<th>Warnings</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Figure 1

```
RXVP80™
Sample STATIC ANALYSIS Report
```
SAMPLE OUTPUT

STATIC ANALYSIS

<table>
<thead>
<tr>
<th>STMT</th>
<th>NEST</th>
<th>LINE</th>
<th>SOURCE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>SUBROUTINE AVER ( Ai; N; ANS )</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>INTEGER N; I; J</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>REAL Ai; ANS, SUM</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>J = 1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>5</td>
<td>WHILE ( I .LE. N )</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>6</td>
<td>J = J + 1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>7</td>
<td>END WHILE</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>8</td>
<td>ANS = SUM/FLOAT(N)</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>9</td>
<td>RETURN</td>
<td></td>
</tr>
</tbody>
</table>

STATIC ANALYSIS SUMMARY

<table>
<thead>
<tr>
<th>ERRORS</th>
<th>WARNINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

FIGURE 2(a)

PROJECT LIBRARY USAGE

Creating a Project Library File

With One Subprogram

G R C SOFTWARE VERIFICATION PROGRAM -- RXVP80

CREATE PROJECT LIBRARY

ENTER COMMANDS FOLLOWED BY END-OF-FILE

EXPAND, OPTION=STATIC.LIST.

FIGURE 2(b)

PROJECT LIBRARY USAGE

Using the Project Library File

Created in Figure 2(a) by Adding a Subprogram
RXVP80 (TM)
SAMPLE OUTPUT

STATEMENT PROFILE
SUBROUTINE SUBA (K1+K2+K3+K4+N)

INTERFACE CHARACTERISTICS

ARGUMENTS 5
COMMON 1
ENTRY 1
EXIT 5
WRITE 12

STATEMENT CHARACTERISTICS

STATEMENT CHARACTERISTICS

DECLARATION...

DATA 3 4.8
COMMON 1 1.6
FORMAT 2 3.2

EXECUTABLE...

DO 1 1.6
END 1 1.6
ASSIGNMENT 11 17.5
GOTO 7 11.1
SUBROUTINE 1 11.6
WRITE 12 19.0
CONTINUE 4 4.3
RETURN 2 3.2

DECISION...

IF 4 6.3
CONTINUE 1 1.6

DOCUMENTATION...

COMMENT 13 20.6

Figure 3(f)
RXVP80™
Sample Local STATEMENT PROFILE Report

INVOCATION SUMMARY

ENTRY LISTS OF CALLS

BKDEMO IS CALLED BY --NONE--
AND CALLS --NONE--
MAIN IS CALLED BY --NONE--
AND CALLS --NONE--
SUBA IS CALLED BY --NONE--
AND CALLS --NONE--

Figure 3(g)
RXVP80™
Sample Global INVOCATION SUMMARY Report

COMMON SUMMARY

COMMON MODULES WHICH INCLUDE THE COMMON

NAMES BKDEMO MAIN SUBA

Figure 3(h)
RXVP80™
Sample Global COMMON SUMMARY Report
## COMMON MATRICES

**LEGEND**

- **C** = FIRST USED IN A CALL
- **E** = EQUIVALENCED
- **S** = SET
- **U** = USED
- **X** = SET AND USED


### FIGURE 3(i)

Sample Global COMMON MATRICES Report

### I/O STATEMENTS

The following modules contain I/O statements:
- **MAIN**
- **SUBA**

### I/O STATEMENTS AND ASSOCIATED FORMATS

--- MAIN ---

<table>
<thead>
<tr>
<th>STATEMENT</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 16 600</td>
<td>WRITE (6,599) ISUBS(1)</td>
</tr>
<tr>
<td>10 19 S99</td>
<td>WRITE (4,99)</td>
</tr>
<tr>
<td>14 22 502</td>
<td>WRITE (5,500) ISUBK(1)</td>
</tr>
<tr>
<td>17 26 500</td>
<td>WRITE (5,500)</td>
</tr>
<tr>
<td>19 28 601</td>
<td>WRITE (1H,16H)</td>
</tr>
</tbody>
</table>

--- SUBA ---

<table>
<thead>
<tr>
<th>STATEMENT</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 9</td>
<td>WRITE (6,599) ISUBS(2)</td>
</tr>
</tbody>
</table>

### FIGURE 3(j)

RXVP80™

Sample Global I/O STATEMENTS Report

### COMMON CROSS REFERENCE

<table>
<thead>
<tr>
<th>NAME</th>
<th>SCOPE</th>
<th>MODULE</th>
<th>USED/SET/EQUIVALENCED</th>
<th>INDICATES SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISUBS</td>
<td>NAMES</td>
<td>BKDEMO</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAMES</td>
<td>MAIN</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAMES</td>
<td>SUBA</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

### FIGURE 3(k)

RXVP80™

Sample Global COMMON CROSS REFERENCE Report
### RXVP80 (TM)

#### SAMPLE OUTPUT

---

**DD-PATH DEFINITIONS**

**SUBROUTINE SUBA ( K1,K2,K3,K4,N )**

<table>
<thead>
<tr>
<th>STMT</th>
<th>NEST</th>
<th>LINE</th>
<th>SOURCE...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>C</td>
<td>SUBROUTINE SUBA ( K1,K2,K3,K4,N )</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>C</td>
<td>COMMON / NAMES / ISUBS(4) # DPATH 1 IS PROCEDURE ENTRY</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>C</td>
<td>DATA L2:L4:L6:L8:L10 / 2HA2:2HA4:2HA6:2HA8:2HA10 / # DPATH 3 IS FALSE BRANCH</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>C</td>
<td>WRITE (6,599) ISUBS(2) # DPATH 4 IS TRUE BRANCH</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>C</td>
<td>WRITE (6,600) L1:L81 # DPATH 6 IS TRUE BRANCH</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>C</td>
<td>WRITE (6,600) L2:L82 # DPATH 7 IS FALSE BRANCH</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>C</td>
<td>WRITE (6,600) L3:L83</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>C</td>
<td>WRITE (6,600) L4:L84</td>
</tr>
<tr>
<td>11</td>
<td>13</td>
<td>C</td>
<td>WRITE (6,600) L5:L85</td>
</tr>
<tr>
<td>12</td>
<td>14</td>
<td>C</td>
<td>WRITE (6,600) L6:L86</td>
</tr>
<tr>
<td>13</td>
<td>15</td>
<td>C</td>
<td>WRITE (6,600) L7:L87</td>
</tr>
<tr>
<td>14</td>
<td>16</td>
<td>C</td>
<td>WRITE (6,600) L8:L88</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>C</td>
<td>WRITE (6,600) L9:L89</td>
</tr>
<tr>
<td>16</td>
<td>18</td>
<td>C</td>
<td>WRITE (6,600) L10:L90</td>
</tr>
<tr>
<td>17</td>
<td>19</td>
<td>C</td>
<td>WRITE (6,600) L11:L91</td>
</tr>
<tr>
<td>18</td>
<td>20</td>
<td>C</td>
<td>WRITE (6,600) L12:L92</td>
</tr>
<tr>
<td>19</td>
<td>21</td>
<td>C</td>
<td>WRITE (6,600) L13:L93</td>
</tr>
<tr>
<td>20</td>
<td>22</td>
<td>C</td>
<td>WRITE (6,600) L14:L94</td>
</tr>
<tr>
<td>21</td>
<td>23</td>
<td>C</td>
<td>WRITE (6,600) L15:L95</td>
</tr>
<tr>
<td>22</td>
<td>24</td>
<td>C</td>
<td>WRITE (6,600) L16:L96</td>
</tr>
<tr>
<td>23</td>
<td>25</td>
<td>C</td>
<td>WRITE (6,600) L17:L97</td>
</tr>
<tr>
<td>24</td>
<td>26</td>
<td>C</td>
<td>WRITE (6,600) L18:L98</td>
</tr>
<tr>
<td>25</td>
<td>27</td>
<td>C</td>
<td>WRITE (6,600) L19:L99</td>
</tr>
<tr>
<td>26</td>
<td>28</td>
<td>C</td>
<td>WRITE (6,600) L20:L100</td>
</tr>
<tr>
<td>27</td>
<td>29</td>
<td>C</td>
<td>WRITE (6,600) L21:L101</td>
</tr>
<tr>
<td>28</td>
<td>30</td>
<td>C</td>
<td>WRITE (6,600) L22:L102</td>
</tr>
<tr>
<td>29</td>
<td>31</td>
<td>C</td>
<td>WRITE (6,600) L23:L103</td>
</tr>
<tr>
<td>30</td>
<td>32</td>
<td>C</td>
<td>WRITE (6,600) L24:L104</td>
</tr>
<tr>
<td>31</td>
<td>33</td>
<td>C</td>
<td>WRITE (6,600) L25:L105</td>
</tr>
<tr>
<td>32</td>
<td>34</td>
<td>C</td>
<td>WRITE (6,600) L26:L106</td>
</tr>
<tr>
<td>33</td>
<td>35</td>
<td>C</td>
<td>WRITE (6,600) L27:L107</td>
</tr>
<tr>
<td>34</td>
<td>36</td>
<td>C</td>
<td>WRITE (6,600) L28:L108</td>
</tr>
<tr>
<td>35</td>
<td>37</td>
<td>C</td>
<td>WRITE (6,600) L29:L109</td>
</tr>
<tr>
<td>36</td>
<td>38</td>
<td>C</td>
<td>WRITE (6,600) L30:L110</td>
</tr>
<tr>
<td>37</td>
<td>39</td>
<td>C</td>
<td>WRITE (6,600) L31:L111</td>
</tr>
<tr>
<td>38</td>
<td>40</td>
<td>C</td>
<td>WRITE (6,600) L32:L112</td>
</tr>
<tr>
<td>39</td>
<td>41</td>
<td>C</td>
<td>WRITE (6,600) L33:L113</td>
</tr>
</tbody>
</table>

---

**FIGURE 4(a)**

Sample *RXVP80™* Report

Showing Where Probes Were Automatically Placed

**RECORD OF DECISION TO DECISION (DD-PATH) EXECUTION**

<table>
<thead>
<tr>
<th>MODULE</th>
<th>SUBRUTINE</th>
<th>TEST CASE NO.</th>
<th>I</th>
<th>I</th>
<th>I</th>
<th>I</th>
<th>I</th>
<th>I</th>
<th>I</th>
<th>I</th>
<th>I</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**FIGURE 4(b)**

Sample DETAILED EXECUTION COVERAGE Report

For Program in Figure 4(a)
### Summary -- This Test

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Module Name</th>
<th>Number of D-D Paths</th>
<th>Number of Invocations Traversed</th>
<th>Per Cent Coverage</th>
<th>Number of Tests Invocations Traversed</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MAIN</td>
<td>7</td>
<td>1</td>
<td>5</td>
<td>71.43</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SUBA</td>
<td>11</td>
<td>6</td>
<td>10</td>
<td>90.91</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>$ALL$</td>
<td>10</td>
<td>15</td>
<td>83.33</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>MAIN</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>57.14</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>SUBA</td>
<td>11</td>
<td>2</td>
<td>11</td>
<td>100.00</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>$ALL$</td>
<td>10</td>
<td>15</td>
<td>83.33</td>
<td>2</td>
<td>17</td>
</tr>
</tbody>
</table>

**Figure 4(c)**

RXVP80™

Sample SUMMARY EXECUTION COVERAGE Report

---

### Cumulative Summary

<table>
<thead>
<tr>
<th>Module</th>
<th>Test</th>
<th>Paths</th>
<th>List of Decision To Decision Paths Not Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td></td>
<td></td>
<td>CUMUL 2 3 5 3 7</td>
</tr>
<tr>
<td>SUBA</td>
<td></td>
<td></td>
<td>CUMUL 0 0 0</td>
</tr>
</tbody>
</table>

**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


CONTACT: WILLIAM R. DEHAAN, GENERAL RESEARCH CORP, 5383 HOLLISTER AVE, PO BOX 6770, SANTA BARBARA, CA, 93111, USA, 805-964-7724
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.

INPUT

- SUBJECT
  - . DATA INPUT.................................................. CS4 ( 1)
  - . DATA INPUT.................................................. PSL/PSA ( 38)
  - . DATA INPUT.................................................. SLIM ( 49)
  - . DATA INPUT.................................................. PWB FOR VAX/VMS ( 64)
  - . DATA INPUT.................................................. TOOLS DATABASE (180)
  - . CODE INPUT.................................................. CS4 ( 1)
  - . CODE INPUT.................................................. LILITH ( 7)
  - . CODE INPUT.................................................. VIRTUAL OS ( 15)
  - . CODE INPUT.................................................. THE ENGINE ( 33)
  - . CODE INPUT.................................................. PWB FOR VAX/VMS ( 64)
  - . CODE INPUT.................................................. AFFIRM ( 69)
  - . CODE INPUT.................................................. ARGUS/MICRO ( 83)
  - . CODE INPUT.................................................. DYN A ( 88)
  - . CODE INPUT.................................................. COMM A P ( 96)
  - . CODE INPUT.................................................. LOGICFLOW (103)
  - . CODE INPUT.................................................. SOFTOOL 80 (TM) (122)
  - . CODE INPUT.................................................. ITB (128)
  - . CODE INPUT.................................................. ISUS (136)
  - . CODE INPUT.................................................. INSTRU (159)
  - . CODE INPUT.................................................. FTN-77 ANALYZER (174)
  - . CODE INPUT.................................................. UCSD P-SYSTEM (194)
  - . CODE INPUT.................................................. MUSEF (197)
  - . CODE INPUT.................................................. IFTRAN (TM) (200)
  - . CODE INPUT.................................................. RXVP80 (TM) (208)
  - . CSL.......................................................... CS4 ( 1)
  - . FORTRAN................................................. ARGUS/MICRO ( 83)
  - . FORTRAN................................................. DYN A ( 88)
  - . FORTRAN................................................. COMM A P ( 96)
  - . FORTRAN................................................. LOGICFLOW (103)
  - . FORTRAN................................................. ITB ( 128)
FORTRAN ISUS (136)
FORTRAN INSTRU (159)
FORTRAN FTN-77 ANALYZER (174)
FORTRAN UCSD P-SYSTEM (194)
FORTRAN IFTRAN (TM) (200)
FORTRAN RXVP80 (TM) (208)
MODULA II LILITH (7)
MODULA LILITH (7)
RATFOR VIRTUAL OS (15)
PASCAL ARGUS/MICRO (83)
PASCAL UCSD P-SYSTEM (194)
COBOL THE ENGINE (33)
FORTRAN 77 ARGUS/MICRO (83)
FORTRAN 77 FTN-77 ANALYZER (174)
FORTRAN 77 UCSD P-SYSTEM (194)
FORTRAN 66 DYNAP (88)
BAL LOGICFLOW (103)
ASSEMBLY LANGUAGE LOGICFLOW (103)
FORTRAN IV (G1) LOGICFLOW (103)
SRTRAN ITB (128)
SRTRAN ISUS (136)
IFTRAN IFTRAN (TM) (200)
IFTRAN RXVP80 (TM) (208)
TEXT INPUT LILITH (7)
TEXT INPUT SARA (10, 76, 163)
TEXT INPUT VIRTUAL OS (15)
TEXT INPUT ARGUS/MICRO (83)
TEXT INPUT SCHEMACODE (152)
TEXT INPUT ONLINE ASSIST (167)
VHLL INPUT SARA (10, 76, 163)
VHLL INPUT SCG (21)
VHLL INPUT DQM (25)
VHLL INPUT AISIM (29)
VHLL INPUT PSL/PSA (38)
VHLL INPUT SDDL (44)
VHLL INPUT POD (58)
VHLL INPUT AFFIRM (69)
VHLL INPUT LOGICFLOW (103)
VHLL INPUT SREM (111)
VHLL INPUT SDP (117)
VHLL INPUT SOFTOOL 80 (TM) (122)
VHLL INPUT FAME (146)
VHLL INPUT SCHEMACODE (152)
VHLL INPUT SRIMP (186)
VHLL INPUT AUTO-DBO (190)
VHLL INPUT IFTRAN (TM) (200)
SL1 SARA (10, 76, 163)
GMF SARA (10, 76, 163)
BNF SARA (10, 76, 163)
DESIGN SPECIFICATION SCG (21)
DESIGN SPECIFICATION DQM (25)
DESIGN SPECIFICATION AISIM (29)
DESIGN SPECIFICATION AUTO-DBO (190)
REQUIREMENTS LANGUAGE PSL/PSA (38)
PSL PSL/PSA (38)
PROBLEM STATEMENT LANGUAGE ...................................... PSL/PSA ( 38)
SDDL ................................................................................. SDDL ( 44)
SYSTEM SPECIFICATION .................................................. POD ( 58)
ALGEBRAIC SPECIFICATIONS ............................................ AFFIRM ( 69)
DESIGN LANGUAGE ........................................................ LOGICFLOW ( 103)
DL ..................................................................................... LOGICFLOW ( 103)
REQUIREMENTS STATEMENT LANGUAGE ......................... SREM ( 111)
RSL ................................................................................... SREM ( 111)
SCHEMATIC PSEUDO CODE .............................................. SCHEMACODE ( 152)
MODEL DESCRIPTION ...................................................... SRIMP ( 186)
IFTRAN .............................................................................. IFTRAN (TM) ( 200)

FUNCTION
TRANSFORMATION
TRANSLATION ........................................................................ CS4 ( 1)
TRANSLATION ...................................................................... LILITH ( 7)
TRANSLATION ..................................................................... SARA ( 10, 76, 163)
TRANSLATION ...................................................................... VIRTUAL OS ( 15)
TRANSLATION ................................................................. SCG ( 21)
TRANSLATION ..................................................................... DQM ( 25)
TRANSLATION ...................................................................... AISIM ( 29)
TRANSLATION ..................................................................... THE ENGINE ( 33)
TRANSLATION ...................................................................... ARGUS/MICRO ( 83)
TRANSLATION ..................................................................... LOGICFLOW ( 103)
TRANSLATION ................................................................. SOFTOOL 80 (TM) ( 122)
TRANSLATION ...................................................................... SRIMP ( 186)
TRANSLATION ................................................................. AUTO-DBO ( 190)
TRANSLATION ..................................................................... UCSD P-SYSTEM ( 194)
TRANSLATION ................................................................. MSEF ( 197)
TRANSLATION ................................................................. IFTRAN (TM) ( 200)
TRANSLATION ................................................................. RXVP80 (TM) ( 208)
EDITING .............................................................................. LILITH ( 7)
EDITING .............................................................................. VIRTUAL OS ( 15)
EDITING .............................................................................. THE ENGINE ( 33)
EDITING .............................................................................. ARGUS/MICRO ( 83)
EDITING .............................................................................. LOGICFLOW ( 103)
EDITING .............................................................................. ISUS ( 136)
EDITING .............................................................................. SCHEMACODE ( 152)
EDITING .............................................................................. SRIMP ( 186)
EDITING .............................................................................. AUTO-DBO ( 190)
EDITING .............................................................................. UCSD P-SYSTEM ( 194)
EDITING .............................................................................. MSEF ( 197)
EDITING .............................................................................. IFTRAN (TM) ( 200)
OPTIMIZATION ................................................................. LILITH ( 7)
RESTRUCTURING .............................................................. SARA ( 10, 76, 163)
RESTRUCTURING .............................................................. SCG ( 21)
RESTRUCTURING .............................................................. THE ENGINE ( 33)
RESTRUCTURING .............................................................. LOGICFLOW ( 103)
RESTRUCTURING .............................................................. SCHEMACODE ( 152)
RESTRUCTURING .............................................................. RXVP80 (TM) ( 208)
FORMATTING ..................................................................... THE ENGINE ( 33)
FORMATTING ..................................................................... SDDL ( 44)
FORMATTING ..................................................................... LOGICFLOW ( 103)
FORMATTING ..................................................................... SDP ( 117)
FORMATTING ...................................................................... SOFTOOL 80 (TM) ( 122)
<table>
<thead>
<tr>
<th>Item</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formatting</td>
<td>SCHEMACODE (152)</td>
</tr>
<tr>
<td>Formatting</td>
<td>SRIMP (186)</td>
</tr>
<tr>
<td>Formatting</td>
<td>IFTRAN (TM) (200)</td>
</tr>
<tr>
<td>Formatting</td>
<td>RXVP80 (TM) (208)</td>
</tr>
<tr>
<td>Instrumentation</td>
<td>THE ENGINE (33)</td>
</tr>
<tr>
<td>Instrumentation</td>
<td>ARGUS/MICRO (83)</td>
</tr>
<tr>
<td>Instrumentation</td>
<td>DYNA (88)</td>
</tr>
<tr>
<td>Instrumentation</td>
<td>SOFTOOL 80 (TM) (122)</td>
</tr>
<tr>
<td>Instrumentation</td>
<td>ITB (128)</td>
</tr>
<tr>
<td>Instrumentation</td>
<td>INSTRU (159)</td>
</tr>
<tr>
<td>Instrumentation</td>
<td>FTN-77 ANALYZER (174)</td>
</tr>
<tr>
<td>Instrumentation</td>
<td>RXVP80 (TM) (208)</td>
</tr>
</tbody>
</table>

**Static Analysis**

<table>
<thead>
<tr>
<th>Item</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>CS4 (1)</td>
</tr>
<tr>
<td>Management</td>
<td>VIRTUAL OS (15)</td>
</tr>
<tr>
<td>Management</td>
<td>PWB FOR VAX/VMS (64)</td>
</tr>
<tr>
<td>Management</td>
<td>ARGUS/MICRO (83)</td>
</tr>
<tr>
<td>Management</td>
<td>SREM (111)</td>
</tr>
<tr>
<td>Management</td>
<td>SOFTOOL 80 (TM) (122)</td>
</tr>
<tr>
<td>Management</td>
<td>ISUS (136)</td>
</tr>
<tr>
<td>Management</td>
<td>SCHEMACODE (152)</td>
</tr>
<tr>
<td>Management</td>
<td>ONLINE ASSIST (167)</td>
</tr>
<tr>
<td>Management</td>
<td>TOOLS DATABASE (180)</td>
</tr>
<tr>
<td>Management</td>
<td>MSEF (197)</td>
</tr>
<tr>
<td>Management</td>
<td>RXVP80 (TM) (208)</td>
</tr>
<tr>
<td>Data Base Management</td>
<td>CS4 (1)</td>
</tr>
<tr>
<td>Data Base Management</td>
<td>SREM (111)</td>
</tr>
<tr>
<td>Files Management</td>
<td>VIRTUAL OS (15)</td>
</tr>
<tr>
<td>Files Management</td>
<td>ARGUS/MICRO (83)</td>
</tr>
<tr>
<td>Configuration Management</td>
<td>PWB FOR VAX/VMS (64)</td>
</tr>
<tr>
<td>Configuration Management</td>
<td>SOFTOOL 80 (TM) (122)</td>
</tr>
<tr>
<td>Change Control</td>
<td>ISUS (136)</td>
</tr>
<tr>
<td>Change Control</td>
<td>MSEF (197)</td>
</tr>
<tr>
<td>Documentation Management</td>
<td>ONLINE ASSIST (167)</td>
</tr>
<tr>
<td>Management Planning</td>
<td>TOOLS DATABASE (180)</td>
</tr>
<tr>
<td>Version Control</td>
<td>MSEF (197)</td>
</tr>
<tr>
<td>Test Data Management</td>
<td>MSEF (197)</td>
</tr>
<tr>
<td>Data Flow Analysis</td>
<td>SARA (10, 76, 163)</td>
</tr>
<tr>
<td>Data Flow Analysis</td>
<td>SOFTOOL 80 (TM) (122)</td>
</tr>
<tr>
<td>Data Flow Analysis</td>
<td>ISUS (136)</td>
</tr>
<tr>
<td>Data Flow Analysis</td>
<td>SRIMP (186)</td>
</tr>
<tr>
<td>Data Flow Analysis</td>
<td>RXVP80 (TM) (208)</td>
</tr>
<tr>
<td>Cross Reference</td>
<td>SARA (10, 76, 163)</td>
</tr>
<tr>
<td>Cross Reference</td>
<td>VIRTUAL OS (15)</td>
</tr>
<tr>
<td>Cross Reference</td>
<td>PSL/PSA (38)</td>
</tr>
<tr>
<td>Cross Reference</td>
<td>COMMAP (96)</td>
</tr>
<tr>
<td>Cross Reference</td>
<td>SDP (117)</td>
</tr>
<tr>
<td>Cross Reference</td>
<td>RXVP80 (TM) (208)</td>
</tr>
<tr>
<td>Completeness Checking</td>
<td>SARA (10, 76, 163)</td>
</tr>
<tr>
<td>Completeness Checking</td>
<td>PSL/PSA (38)</td>
</tr>
<tr>
<td>Completeness Checking</td>
<td>PWB FOR VAX/VMS (64)</td>
</tr>
<tr>
<td>Completeness Checking</td>
<td>SREM (111)</td>
</tr>
<tr>
<td>Completeness Checking</td>
<td>SOFTOOL 80 (TM) (122)</td>
</tr>
</tbody>
</table>
. . COMPLETENESS CHECKING. . AUTO-DBO (190)
. . COMPLETENESS CHECKING. RXVP80(TM) (208)
. . CONSISTENCY CHECKING. SARA (10, 76, 163)
. . CONSISTENCY CHECKING. PSL/PSA (38)
. . CONSISTENCY CHECKING. AFFIRM (69)
. . CONSISTENCY CHECKING. SREM (111)
. . CONSISTENCY CHECKING. SRIMP (186)
. . CONSISTENCY CHECKING. AUTO-DBO (190)
. . CONSISTENCY CHECKING. RXVP80(TM) (208)
. . STRUCTURE CHECKING. SARA (10, 76, 163)
. . STRUCTURE CHECKING. THE ENGINE (33)
. . STRUCTURE CHECKING. LOGICFLOW (103)
. . STRUCTURE CHECKING. SDP (117)
. . STRUCTURE CHECKING. SOFTOOL 80(TM) (122)
. . STRUCTURE CHECKING. ISUS (136)
. . STRUCTURE CHECKING. IFTRAN(TM) (200)
. . STRUCTURE CHECKING. RXVP80(TM) (208)
. . SCANNING. . SARA (10, 76, 163)
. . SCANNING. . SDDL (44)
. . SCANNING. . SDP (117)
. . SCANNING. . SOFTOOL 80(TM) (122)
. . SCANNING. . RXVP80(TM) (208)
. . COMPARISON. . VIRTUAL OS (15)
. . COMPARISON. . PWB FOR VAX/VMS (64)
. . COMPARISON. . SOFTOOL 80(TM) (122)
. . COMPARISON. . MSEF (197)
. . COMPLEXITY MEASUREMENT. . DQM (25)
. . COMPLEXITY MEASUREMENT. . LOGICFLOW (103)
. . COMPLEXITY MEASUREMENT. . SOFTOOL 80(TM) (122)
. . COMPLEXITY MEASUREMENT. . SCHEMACODE (152)
. . COMPLEXITY MEASUREMENT. . RXVP80(TM) (208)
. . ERROR CHECKING. . THE ENGINE (33)
. . ERROR CHECKING. . PSL/PSA (38)
. . ERROR CHECKING. . COMMAP (96)
. . ERROR CHECKING. . LOGICFLOW (103)
. . ERROR CHECKING. . ISUS (136)
. . ERROR CHECKING. . FAME (146)
. . ERROR CHECKING. . RXVP80(TM) (208)
. . COST ESTIMATION. . SLIM (49)
. . SCHEDULING. . SLIM (49)
. . TIME SCHEDULING. . SLIM (49)
. . PERSONNEL SCHEDULING. . SLIM (49)
. . TYPE ANALYSIS. . AFFIRM (69)
. . TYPE ANALYSIS. . RXVP80(TM) (208)
. . AUDITING. . LOGICFLOW (103)
. . AUDITING. . SOFTOOL 80(TM) (122)
. . INTERFACE ANALYSIS. . SOFTOOL 80(TM) (122)
. . INTERFACE ANALYSIS. . AUTO-DBO (190)
. . INTERFACE ANALYSIS. . RXVP80(TM) (208)
. . STATISTICAL ANALYSIS. . FTN-77 ANALYZER (174)
. . STATISTICAL ANALYSIS. . RXVP80(TM) (208)
. . UNITS ANALYSIS. . RXVP80(TM) (208)
. DYNAMIC ANALYSIS
. . SIMULATION. . SARA (10, 76, 163)
. . SIMULATION. . AISIM (29)
. SIMULATION. ........................................... SLIM ( 49)
. SIMULATION. ........................................... POD ( 58)
. SIMULATION. ........................................... SREM ( 111)
. COVERAGE ANALYSIS ................................... THE ENGINE ( 33)
. COVERAGE ANALYSIS .................................... ARGUS/MICRO ( 83)
. COVERAGE ANALYSIS .................................... DYN ( 88)
. COVERAGE ANALYSIS .................................... SOFTOOL 80 (TM) ( 122)
. COVERAGE ANALYSIS .................................... ITB ( 128)
. COVERAGE ANALYSIS .................................... FTN-77 ANALYZER ( 174)
. COVERAGE ANALYSIS .................................... IFTRAN (TM) ( 200)
. TRACING .................................................. THE ENGINE ( 33)
. TRACING .................................................. SOFTOOL 80 (TM) ( 122)
. TRACING .................................................. ITB ( 128)
. TRACING .................................................. INSTR ( 159)
. TRACING .................................................. FTN-77 ANALYZER ( 174)
. TRACING .................................................. IFTRAN (TM) ( 200)
. TRACING .................................................. RXVP80 (TM) ( 208)
. DATA FLOW TRACING ................................... INSTR ( 159)
. LINEAR PROGRAMMING ................................ SLIM ( 49)
. TUNING ................................................... SOFTOOL 80 (TM) ( 122)
. TUNING ................................................... FTN-77 ANALYZER ( 174)
. TUNING ................................................... IFTRAN (TM) ( 200)
. TUNING ................................................... RXVP80 (TM) ( 208)
. ASSERTION CHECKING ................................ AFFIRM ( 69)
. ASSERTION CHECKING ................................ FTN-77 ANALYZER ( 174)
. ASSERTION CHECKING ................................ IFTRAN (TM) ( 200)
. ASSERTION CHECKING ................................ RXVP80 (TM) ( 208)
. TIMING .................................................. SOFTOOL 80 (TM) ( 122)
. SYMBOLIC EXECUTION ................................ RXVP80 (TM) ( 208)
. CONSTRAINT ANALYSIS ................................. RXVP80 (TM) ( 208)

OUTPUT

USER OUTPUT

. USER-ORIENTED TEXT .................................. CS4 ( 1)
. USER-ORIENTED TEXT .................................. SARA ( 10, 76, 163)
. USER-ORIENTED TEXT .................................. VIRTUAL OS ( 15)
. USER-ORIENTED TEXT .................................. ARGUS/MICRO ( 83)
. USER-ORIENTED TEXT .................................. SREM ( 111)
. USER-ORIENTED TEXT .................................. SDP ( 117)
. USER-ORIENTED TEXT .................................. SOFTOOL 80 (TM) ( 122)
. USER-ORIENTED TEXT .................................. ONLINE ASSIST ( 167)
. USER-ORIENTED TEXT .................................. AUTO-DBO ( 190)
. DOCUMENTATION ....................................... CS4 ( 1)
. DOCUMENTATION ....................................... SARA ( 10, 76, 163)
. DOCUMENTATION ....................................... SDP ( 117)
. DOCUMENTATION ....................................... SOFTOOL 80 (TM) ( 122)
. DOCUMENTATION ....................................... ONLINE ASSIST ( 167)
. DOCUMENTATION ....................................... AUTO-DBO ( 190)
. ON-LINE ASSISTANCE .................................. SARA ( 10, 76, 163)
. REPORTS ................................................ SREM ( 111)
. LISTINGS ............................................... CS4 ( 1)
. LISTINGS ............................................... LILITH ( 7)
. LISTINGS ............................................... SARA ( 10, 76, 163)
. LISTINGS ............................................... VIRTUAL OS ( 15)
LISTINGS

THE ENGINE (33)

PSL/PSA (38)

PSL/PSA (44)

AFFIRM (69)

ARGUS/MICRO (83)

DYNA (88)

LOGICFLOW (103)

SREMY (111)

SDP (117)

SOFTOOL 80 (TM) (122)

ITB (128)

ISUS (136)

SCHEMACODE (152)

INSTRU (159)

FTN-77 ANALYZER (174)

IFTRAN (186)

UCSD P-SYSTEM (194)

MSEF (197)

IFTRAN (TM) (200)

RXVP80 (TM) (208)

LILITH (7)

SARA (10, 76, 163)

SCG (21)

DQM (25)

AISIM (29)

PSL/PSA (38)

SLIM (49)

POD (58)

ARGUS/MICRO (83)

LOGICFLOW (103)

SREMY (111)

SDP (117)

FAME (146)

SRIMP (186)

IFTRAN (TM) (200)

RXVP80 (TM) (208)

STRUCTURE CHARTS

SCG (21)

DQM (25)

LOGICFLOW (103)

BAR CHARTS

SLIM (49)

LINE GRAPHS

SLIM (49)

FLOW CHARTS

LOGICFLOW (103)

HIERARCHICAL TREE

SDP (117)

FAME (146)

HIERARCHICAL TREE

SRIMP (186)

CONTROL MAP

FAME (146)

ACTIVITY DIAGRAM

SRIMP (186)

VIRTUAL OS (15)

PSL/PSA (38)

PWB FOR VAX/VMS (64)

AFFIRM (69)

ARGUS/MICRO (83)

COMMAPPING (96)

SREMY (111)

SOFTOOL 80 (TM) (122)
• DIAGNOSTICS ......................................................... ISUS (136)
• DIAGNOSTICS ......................................................... FAME (146)
• DIAGNOSTICS ......................................................... IFTRAN (TM) (200)
• DIAGNOSTICS ......................................................... RXVP80 (TM) (208)
• TABLES ................................................................. DQM (25)
• TABLES ................................................................. AISIM (29)
• TABLES ................................................................. PSL/PSA (38)
• TABLES ................................................................. SLIM (49)
• TABLES ................................................................. POD (58)
• TABLES ................................................................. PWB FOR VAX/VMS (64)
• TABLES ................................................................. ARGUS/MICRO (83)
• TABLES ................................................................. DYNA (88)
• TABLES ................................................................. COMMAP (96)
• TABLES ................................................................. SOFTOOL 80 (TM) (122)
• TABLES ................................................................. ITB (128)
• TABLES ................................................................. INSTRU (159)
• TABLES ................................................................. FTN-77 ANALYZER (174)
• TABLES ................................................................. TOOLS DATABASE (180)
• TABLES ................................................................. RXVP80 (TM) (208)

MACHINE OUTPUT
• OBJECT CODE OUTPUT ................................................ .CS4 (1)
• OBJECT CODE OUTPUT ................................................ LILITH (7)
• OBJECT CODE OUTPUT ................................................ .CS4 (1)
• DATA OUTPUT .......................................................... SARA (10, 76, 163)
• DATA OUTPUT .......................................................... PSL/PSA (38)
• DATA OUTPUT .......................................................... PWB FOR VAX/VMS (64)
• SOURCE CODE OUTPUT ................................................ SARA (10, 76, 163)
• SOURCE CODE OUTPUT ................................................ VIRTUAL OS (15)
• SOURCE CODE OUTPUT ................................................ THE ENGINE (33)
• SOURCE CODE OUTPUT ................................................ DYNA (88)
• SOURCE CODE OUTPUT ................................................ LOGICFLOW (103)
• SOURCE CODE OUTPUT ................................................ SOFTOOL 80 (TM) (122)
• SOURCE CODE OUTPUT ................................................ ITB (128)
• SOURCE CODE OUTPUT ................................................ ISUS (136)
• SOURCE CODE OUTPUT ................................................ SCHEMACODE (152)
• SOURCE CODE OUTPUT ................................................ INSTRU (159)
• SOURCE CODE OUTPUT ................................................ FTN-77 ANALYZER (174)
• SOURCE CODE OUTPUT ................................................ IFTRAN (TM) (200)
• SOURCE CODE OUTPUT ................................................ RXVP80 (TM) (208)
• FORTRAN ............................................................... VIRTUAL OS (15)
• FORTRAN ............................................................... DYNA (88)
• FORTRAN ............................................................... LOGICFLOW (103)
• FORTRAN ............................................................... ITB (128)
• FORTRAN ............................................................... ISUS (136)
• FORTRAN ............................................................... SCHEMACODE (152)
• FORTRAN ............................................................... INSTRU (159)
• FORTRAN ............................................................... IFTRAN (TM) (200)
• FORTRAN ............................................................... RXVP80 (TM) (208)
• COBOL ................................................................. THE ENGINE (33)
• FORTRAN 77 ............................................................ FTN-77 ANALYZER (174)
• FORTRAN 66 ............................................................ DYNA (88)
• JOVIAL ................................................................. LOGICFLOW (103)
• SRTRAN ............................................................... ITB (128)
• SRTRAN ............................................................... ISUS (136)
IFTRAN
PROMPTS
VHLL OUTPUT
VHLL OUTPUT
DESIGN SPECIFICATION
PSL
INTERMEDIATE CODE

RXVP80 (TM) (208)
SARA (10, 76, 163)
SRIMP (186)
AUTO-DBO (190)
AUTO-DBO (190)
SRIMP (186)
MSEF (197)
### Bibilographic Data Sheet

1. **Publication or Report No.** NBS SP 500-80
2. **Performing Organ. Report No.**
3. **Publication Date.** October 1981

### 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

### Author(s)

Raymond C. Houghton, Jr., Editor

### Performing Organization

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

### Sponsor's Name and Complete Address

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

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

### Supplementary Notes

Library of Congress Catalog Card Number: 81-600109

Document describes a computer program; SF-185, FIPS Software Summary, is attached.

### 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.

### Availability

- Unlimited
- For Official Distribution, Do Not Release to NTIS
- Order From National Technical Information Service (NTIS), Springfield, VA. 22161

### No. of Printed Pages

238

### Price

$6.50
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 20036.

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 following NBS publications—FIPS and NBSIR’s—from the National Technical Information Services, Springfield, VA 22161.


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.