ISRAEL AIRCRAFT INDUSTRIES LTD
ENGINEERING DIVISION

עותק מס' _____ מתוך _____

הועתן _____

SECURITY CLASS UNCLASSIFIED

COML CLASS

A	COST CENT	R	PROJ DATE	WORK ORDER	EFFECT ON	REFERENCE		EO NO	CARD
1	4540		13\07\81	8149053-4				001	x10
26	27—30	31	32—37	38—45	46—55	56—73	74	75—77	78—80

DOCUMENT IDENTIFIER

CAT	PREFIX	PROJECT	CASC	SEQUENCE NO
TR		ENG	4540	81107/18

SUBJECT נושא

SURVEY OF U.S. SOFTWARE TOOLS AND TECHNOLOGY - SUMMARY OF THE MAY 11 - JUNE 11 VISITS

תפוצה:

מר י. אביב - אפ"א/אלת"א	(1)	מר ע. ה"ירלי - מח' 4610	(1)
מר א. אופטובסקי - מח' 4170	(1)	מר ג. טריפון - מח' 4610	(1)
מר ש. אטאלי - מח' 4100	(1)	מר י. כהן - מח' 4890	(1)
מר ר. אמארו - מח' 4920	(1)	ד"ר י. לאב - מו"פ/תע"א	(1)
מר ג. אפרים - מח' 4710	(1)	מר ע. לוטן - מח' 4670	(1)
מר מ. בונס - מח' 4680	(1)	ד"ר ב. לוז - אלת"א	(1)
מר מ. בלומקין - מח' 4100	(1)	מר י. מעין - מח' 4920	(1)
ד"ר ד. בן-דוד - מו"פ/תע"א	(1)	מר א. סומך - מח' 4751	(1)
ד"ר מ. בסיל - מח' 4711	(1)	מר ס. סמואל - מח' 4500	(1)
מר מ. ברימן - מח' 4100	(1)	מר י. סעד - מח' 4711	(1)
מר י. ברכה - מח' 4890	(1)	מר ד. סער - מח' 4100	(1)
מר א. בר-שוע - אפ"א/מכ"ת	(1)	מר ח. עשבי - ממ"ן	(1)
מר א. גולוסובסקר - מח' 4920	(1)	גב' ר. צוקן - ממ"ן	(1)
מר א. גמזו - מח' 4920	(1)	מר א. קצ'קו - מכ"ת	(1)
מר ש. גסול - מח' 4360	(1)	מר ד. רביב - מלח"ט	(1)
ד"ר מ. דביר - מו"פ/תע"א	(1)	מר ח. רוט - אפ"א/מכ"ת	(1)
מר א. דוד - מח' 4920	(1)	מר ר. רוסטל - מח' 4871	(1)
ד"ר ב. הלפרין - תמ"מ	(1)	מר מ. רשף - מכ"ת	(1)
מר ב. הניגאל - מח' 4610	(1)	מר י. שוחט - מח' 4890	(1)
מר ש. הרלב - תמ"מ	(1)	מר ש. שחף - מח' 9455	(1)
ד"ר ד. הורי - מח' 4620	(1)	מר י. שרון - מח' 4670	(1)
		מחלקה 4540	(5)

		EO	DOC	COVER					
					Author	Dr. Ben Livson		4540	7.6.81
					Check	Dr. Ben Livson		4540	7.6.81
					Dept	Daniel Grouchko		4540	7.6.81
Name/Code	Dept				Project	Sergio Samuel		4500	7.6.81
DISTRIBUTION					APPR	NAME	SIGNATURE	DEPT	DATE

1/6035-D

התעשיה האוירית לישראל בע"מ
חטיבת הנדסה

4540/810988
DOCUMENT NO. מסמך מס'

UNCLAS
סיווג בטחוני SECURITY CLASS
סיווג מסחרי COMML CLASS

DOCUMENT ISSUE

ISSUE	New															
DATE	7.6.81															

PAGE No.	REPORT PAGES															PAGE SIZE
1																
2																
3																
4																
5																
6																
7																
8																
9																
10																
11																
12																
13																
14																
15																
16																
17																
18																
19																
20																
21																
22																
23																
24																
25																
26																
27																
28																
29																
30																

ADDED PAGES

3/6268 - D

DOCUMENT ISSUE

ISSUE	New															
DATE																

PAGE No.	REPORT PAGES	PAGE SIZE
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
1		
2		
3		
4		
5		
6		
7		
8		
9		
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
0		

ADDED PAGES

CONTENTS

0. SUMMARY & CONCLUSIONS

IAI Engineering Division has a need for an environment of integrated software tools for both general and real-time software development. Some 800 software tools were catalogued, and classified into tens of groups by various U.S. organizations, mainly by aerospace companies. To obtain updated and first hand information, B. Livson visited 17 U.S. organizations during May 11 - June 11: The mix was 40% private companies and 60% academic and other organizations. Three major trends in U.S. software tools and technology were surveyd more closely:

1. Several hundred installations all over the U.S. are using software development work benches with UNIX software tools and a "SHELL" type command language interpreter allowing pipelining of software tools, see Appendix. The UNIX itself had the draw-back of forcing VAX users to give up DEC's VMS operating system. The need arose to provide a UNIX like environment residing on top of VAX VMS or, after a moderate systems programming effort, on top of any operating system of at least 32-bit word length main frames.
   Lawrence Livermoore Laboratories developed and distributed the Berkeley software tools package and SHELL command language interpreter with capabilities comparable to level 7 UNIX.
   B. Livson received full documentation and magnetic tapes to install the Berkeley system on IAI Engineering VAX machines and, after some systems programming, on our CDC CYBER. Berkeley tools package resembles IAI CAD/CDC utilities installed by our C.A.D. Department. However, the powerful SHELL command language interpreter combines the tools together, and is a tremendous help to programmers, see Appendix. For our VAX users, a combined VMS and Berkeley tool environment will be a great boost in productivity, as Hughes Aircraft Company experience indicates.

2. Packages of both static and dynamic testing tools are being developed and/or marketed by TRW, GRC (software workshop), Boeing (IVTS and ARGUS), Colorado University (TOOLPACK), etc. These tool packages usually augment preprocessors from structured FORTRAN into regular FORTRAN, and from ugly FORTRAN into structured FORTRAN.

3/6222-17

Most aerospace and defence contractors are FORTRAN-LANDS, so that such preprocessors will provide an attractive enhancement of their programming environment. Experiences with modern languages such as JOVIAL 73 and PASCAL have not been encouraging. The lack of tools environment, and a large number of bugs even in such basic tools as compilers have handicaped any wider use of modern programming languages. ADA is still at least 2-3 years away from wider application. In the meantime, BELL C-language is most promising allowing UNIX or VMS + EUNICE ≅ UNIX. Most of the testing tools packages have very similar capabilities. The latest GRC software workshop seems to be the best package in its category (2) by being truly interactive, and menu-driven in an "intelligent" way.
IAI Engineering Division should seriously consider acquiring or developing a test tools package.

3. Boeing, TRW and NASA have very ambitious projects to imbed most software development including requirements, design, data dictionaries, documentation, coding, and testing analysis in relational data bases. Although these projects are several years from completion, IAI should very carefully watch these developments, and start its own activities. In general, our technological level seems to match U.S. software development level. In B. Livson's opinion, IAI Engineering Division has the same capability of developing its own software tools as any U.S. aerospace company division.

1. ROME AIR DEVELOPMENT CENTER (RADC)

Griffis Air-Force Base, N.Y. 13441, May 11-12.
Arrangements: U.S.A.F. Office of Scientific Research, European Office of Aerospace R & D, Major James R. Kreer.
RADC contact: Rocco F. Iuorno.
AGENDA:

1.1 Software Requirements/Specifications

RADC expert: William Rzepka.

Desciption: RADC is sponsoring TRW to develop the SREM software requirement tool, and the associated methodology for input preparation. SREM is based on older version 3 of the ISDOS project tool PSA, problem statement analyzer.

3/6222-D

The objective is to release SREM for general use in the U.S.A.F. projects within two years. The emphasis is on:

A. Functional Specifications

W. Rzepka and B. Livson went over a SREM specification for energy management and engine control software of a transport plane. The presentation seemed to be almost identical to IAI PSA trials.

B. Path Specifications

SREM methodology is geared to the presentation of actual execution paths somewhat the same way as very high level flow charts. This seems to contradict the basic notion that a requirement specification should only answer the "what" and not the "how". It seems that SREM is intended as a design tool, too.

C. Performance Requirements

W. Rzepka claimed that SREM is more powerful than PSA in presenting performance requirements such as timing, memory, etc. constraints. Rzepka did not, however, have any updated knowledge on the newer PSA versions.

D. Tool Flexibility

SREM allows the user to define key-words. This does not seem to be any major advantage over PSA, which has an extensive range of key-words. One can always manupulate the set of key-words, since only a small subset is in actual usage. Summarizing, RADC does not have any edge over IAI in software requirements engineering.

1.2 National Software Works (NSW) and the Associated Computer Facility Tour

RADC experts: Dennis Maynard & Patricia Baskinger.

Description: Some fifty computer facilities and thousands of users throughout the United States are connected to the NSW ARPANET. This huge network allows users to access a number of mainframe types / operating systems / software tools.

The main difficulty was not in creating a fast, reroutable and reliable communications network, but an extensive system programming needed to enable different mainframe types to swap files. It would be most vital to network the IAI Maman, Engineering Div, and Mabat IBM, CDC & VAX machines. A similar effort was started in Boeing Computer Services Company.

1.3 Ada Programming Language Status

RADC expert: Elizabeth Kean.

Description: RADC is evaluating U.S.A.F. ADA development projects, and competes with the Army to issue the U.S. DoD ADA compiler and the associated tool package including data base manager, debugger, editor etc. The progress with ADA seems to be much slower than the fuss created about it. RADC did not have even an experimental compiler at the time of B. Livson's visit. Serious problems with ADA's scheduling and tasking mechanisms were encountered.
The general complaint was that ADA is too big. RADC seems to be surprisingly poorly informed about the commercial ADA compiler projects for Intel i APX-432 and VAX 11/780. B. Livson's impression is that ADA is too immature to be applied in our avionics software projects.

1.4 FORTRAN Automated Verification System (FAVS)

RADC expert: Frank La Monica

Description: General Research Corporation has developed automated verification systems for a number of computer languages:
FORTRAN (FAVS), COBOL (CAVS) & JOVIAL (JAVS).
An automated verification system includes both static and dynamic testing tools generating data bases for browsing. (Browsing = data base quering interactively). FAVS includes drivers to execute any combinations of tools and test cases. It instruments the code to collect statistics about paths (un) covered, statements executed, modules invoked, etc. A listing of paths uncovered aids test case generation. In addition, FAVS produces extensive documentation of program interfaces. IAI Engineering Division has tools such as DAVE & PROMO that partly cover FAVS activities, but FAVS has the advantage of being a software development work-bench and not just separate tools.

GRC provides FAVS on a commercial bases for both CDC and VAX 11/780 machines. RADC people have actively used FAVS with excellent cost effectiveness. GRC develops AVS frontends enabling development on any special purpose real-time command and control language. Rewriting AVS is also possible, but a lot costlier than frontend acquisition, according to F. La Monica.

As a highly appealing feature, FAVS includes a preprocessor from FORTRAN into a block structured language, IFTRAN, and another preprocessor from IFTRAN into FORTRAN. In fact, FAVS is written in IFTRAN and is bootstrapped by the preprocessor into FORTRAN for compilation. Thus, it is possible to write in a truly structured language (FORTRAN 77 is not fully structured). It is also possible to automatically transform our old unstructured programs into structured and indented ones. This is a great help in program maintenance.

1.5 Program Support Library (PSL)

RADC expert: Lawrence Lombardo.

Description: PSL is intended for very large software houses, and has been used in the U.S. strategic Air Force software project with 2,000 programmers. The basic idea is an ironman software configuration control, where a programmer cannot upgrade the status of his programming unit without the approval of his manager. Various programming standards and test requirements are automatically monitored. Each program unit must be documented according to a list of key-words. Chief programmers obtain managerial information with any desired combination of key-words over all programmers. PSL is thus a super files search tool. PSL is also used to drive any combination of tools and test cases. Several PSL libraries can communicate with each other.

The practical experience with PSL has been quite poor, however. In fact, RADC has been forced to pay for the overhead of the organization using this tool. Most of the programmers involved just hate this tool. At the moment no PSL version is available for our CDC or VAX machines. In any case, the RADC program support library does not seem to be suitable for the Engineering Division

environment. IBM has sponsored development of the PSL, and perhaps this tool would be useful for Maman.

1.6 RADC Paperless Laboratory

RADC experts: As in 1.2 & 1.5.
Description: RADC paperless laboratory is closely related to the ARPANET and PSL projects. Both RADC and Boeing Computer Services people told about a slow ten year shift towards a paperless environment. The corner-stones of this effort are computer networks, massive disk space, fast communications, personal CRT scopes with large buffers for fast page refresh and a lot of graphics.
Present day technologies are sufficient for a paperless environment, but the investments and training needed necessitate a slow transition.

1.7 System Reliability

RADC experts: Anthony Coppola and Gerald Clion.
Description: An effort to combine hardware and software reliability models is under way. B. Livson remained highly sceptical about RADC reliability modelling efforts. Both N. Singpurwalla and B. Littlewood from the George Washington University claimed that the RADC models are not worth the paper on which they are written. The basic problem with software reliability models is that there is no way to quantify software reliability in the requirements and design phases so that no meaningful software reliability allocation is possible.
There are no software reliability data-banks (excluding the DACS trial that miserably failed) unlike the extensive hardware component data-banks. Hardware concepts such as redundancy are not practical in software reliability due to integration problems, memory and execution time constraints, lack of software development manpower, etc. Any attempt of a full scale redundancy would probably lead to a worse software. Certain methodologies were developed to implement fault tolerant software as deeply nested error routines for critical software components. This, however, does not render to any quantification. As mentioned, software reliability modes can be applied only at the dynamic testing stage.

B. Livson has an idea how to enable the use of these models already at static testing, based on the new automated static analysis tools that extract error information covering some 20% of coding errors, and just 6-8% of all software errors. This, however, gives some idea about error rate per source statement. The main use of statistical models would be to estimate the testing required. This does not work out for the following two reasons:

1. Programmers do everything to obstruct error data collection.
2. The models stabilize only after 75% debugging, and thus render themselves almost useless.

In short, RADC people are frustrated about modelling.
The RADC expert on modelling, Alan Sukert, vacated his position and very little further modelling is under way in RADC.
Clearly, the engineering, and not the statistics, approach should be employed to improve software reliability without trying to quantify the actual reliability attained.
Joseph Cavano in RADC is trying to build comprehensive software quality metrics including numerous areas from portability to readability. Good complexity models would be useful to properly concentrate testing efforts. One must, however, distinguish between problem complexity and programming complexity. Automated aids are available to measure the less important programming complexity.
In general, the complexity models as well as the rest of quantitative software models are in a very poor shape.
RADC has been actively sponsoring the modelling of hardware reliability growth. Since the 1975 Hughes report, very little concrete was achieved.
A. Coppola & G. Clion seem to be quite optimistic about their ability to predict hardware reliability. They spoke about a ±25% accuracy. This does not conform to our experience. It is pointed out that the Coppola group is not involved in any reliability improvement work (Fault Tree or Failure Modes and Effects Analysis) like IAI Reliability Engineering. In B. Livson's opinion, our Department of Statistical Reliability is more practically oriented in our emphasis on confidence levels, K-factors, and failure reporting and corrective action analyses with extensive software development in

these fields. The Coppola group (28 employees) is active in components engineering and sponsors MIL-HDBK-217C updating.

1.8 Data and Analysis Center for Software (DACS)

RADC experts: L. Duval. S. Gloss-Soler, G. Caren.

Description: DACS has sponsored a number of reports:

- Quantitative software models including software life-cycle cost models (not promissing).
- Productivity data-base (of very little value).
- Reliability data-base (no value).

B. Livson has studied these reports in 79-80. DACS is conducting data searches for the industry. DACS wants to use the NBS software tools data-base to counsel private companies on what tools to use. This seems strange to B. Livson, since DACS people are librarians without any knowledge in computer sciences/software engineering.
A lot of fuss was made about the Reagan administration directive to curb information flow to foreign countries, including NATO countries. It turned out that B. Livson is both in the RADC and DACS mailing lists as well as certain United Nations organizations, so that B. Livson, Moscow & Peking until now have received whatever RADC reports in their interest for free...
The whole fuss about information flow curb tends to indicate that RADC has lost its lead in most fields related to software technology. RADC nowdays does very little in-house research and just sponsors work conducted in California based software houses. The atmosphere in RADC seems to be quite depressed and several people did not (could not) hide their feeling that things are not what they used to be. Even in the RADC strong areas, the ARPANET and the general nanomachine and SLIM processor for emulating microprocessors, there seems to be a decline.
The following two RADC bibliographies were given to B. Livson:

- RADC Software Technology Reports (some 400).
- RADC Reliability and Maintainability Reports (some 200).

2. NATIONAL ACADEMY OF SCIENCES,
COMPUTER SCIENCE TECHNOLOGY BOARD
Washington DC, May 18.
Contacts: F. Frischman & J. Blackburn.
Description: R & D policies and "fingerpointing" for further professional contacts, see 3 & 4.

3. NATIONAL SCIENCE FOUNDATION (NSC)
Washington DC, May 18.
Software Engineering: Bruce H. Barnes, Head of Computer Science Section.
Description: NSC sponsors hundreds of research projects in theoretical computer science, software systems science, computer systems design, software engineering, intelligent systems, experimental computer science and equipment, special projects. A list of project abstracts was given to B. Livson. NSC experts stressed that advances in software technology are mainly made by universities with "small" money and not by the big corporations. Of particular interest to IAI Engineering Division is a group of projects called TOOLPACK with the objective of a comprehensive software development workbench. A "state of the project" report was given to B. Livson, outlining the philosophy and architecture of the project.
TOOLPACK is designed for a scientific/engineering environment and for relatively small programming teams. It combines the best ideas of software support libraries and automated verification systems.

Special Projects: Richard Adrion.
Description: Richard Adrion stated that basically nothing more advanced is available in software requirements engineering than our PSA with proper input preparation technologies. A breakthrough is possible within our life-time: Software requirement language and analyzer that generates executable code, obliterating design and coding phases. Bob Balzer from the USC - Info SC. Institute and Noah Prywes from the University of Pennsylvania have developed the first trial tools. Their tools have actually generated themselves from their own requirements specifications, given of course, a certain minimal kernel. Other people, including Daniel Berry from UCLA, do not think that it will ever be possible to generate executable codes from "real" requirements.

4. OFFICE OF NAVAL RESEARCH

Washington D.C., May 18

Contact: Bob Grafton

Description: The main software technology project sponsored is MUTATION analysis for formal testing. The experts in this field are: Richard Demillo, Georgia Institute of Technology (COBOL based automated tool), Fredric Seyword, Yale University (FORTRAN based automated tool).

5. GEORCE WASHINGTON UNIVERSITY
DEPT. OF OPERATIONS RESEARCH

May 18.

System Reliability Models: Nozer D. Singpurwalla.

Description: Extensive work in reliability growth. Copies of recent studies were given to B. Livson. Of particular interest is the possibility to separate between screening and design improvements. Singpurwalla does not believe in statistical software reliability or even in the integrity of most software reliability modelling. Although a statistician, he stressed that in software reliability only an engineering approach pays. Singpurwalla gave an interesting research paper, where he and the Israeli Landberg show how a number of software reliability models can be unified via a Bayesian model with different priors. They show why the present models may yield absurd results by being based on minimum likelihood estimates of the initial number of software errors. The weakness of these models is that they overlook the basic requirement of sampling models, namely the use of a stopping rule.

Software Reliability: Bev Littlewood

Description: Littlewood has been working on software reliability models since 1973, and is perhaps the leading name in this field. It was surprising to notice he is a pure mathematician/statistician with little or no understanding of software engineering/computer sciences. He claimed that one of his models was applied in the F-111 avionics software project. This turned out to be a post mortem of no value... Littlewood could not give any explanation of how one can reach a given quantitative software goal, thus strengthened B. Livson's opinion that quantitative reliability goals are "day dreaming". Littlewood seemed optimistic about other quantitative software models too:

- Putnam software life-cycle cost model is impressive in his opinion (according to RADC miserable, and in B. Livson's opinion corrupt in having a state of technology constant than can be manipulated in-flight or post-mortem to obtain any result).
- McCabe and Basili complexity metrics and automated tools are impressive in Littlewoods opinion. We shall check in to this, although RADC people hold a diametrically opposed opinion.

6. BOEING COMPUTER SERVICES (BCS)

7980 Gallows Court, Vienna, Virginia, May 19.

Outline of BCS activities: Sy Listman, Tech. Mgr. of BCS East, and Mary Dawson, Mgr. of Products.

Description: BCS has a massive computer network with 3,000-4,000 users all over the United States. Its network is more primitive than that of RADC - ARPANET in the sense that files cannot be swapped between different mainframe types (Boeing IBM & CDC hosts), but BCS is working hard on this. BCS has next to ceased from being a software house for outside customers due to heavy losses. BCS people neither used any software cost model of their own nor are they familiar with the Putnam and RCA S-models. The main problem with software life-cycle cost models is the weak link between the model and the actual software work breakdown structure. BCS ignorance of these models was, however, surprising. BCS EAST is the "Maman" of Boeing today.

Advanced Technology and Development (ATAD): Bruce Wilson.

Description: A lot of high level R & D manager talk about software technology. For instance, Wilson enthusiastically described how BCS measures software reliability with the Markov model. Even the father of this model, Bev Littlewood, would not claim that this model had any practical applicability.

Software Quality Assurance: Arthur Russel and Julia Beck.

Description: BCS EAST has a four-man SQA group. None of these people is a professional in computer sciences/software engineering. This group is mainly involved in administrative type of work: software standards and procedures implementation, preparation of check-lists showing correspondence between software requirements - design - coding - testing items or components.

Their checks are of entirely mechanical nature, however. Their approach is nevertheless practical, and by cataloguing the test cases a lot of effort is eliminated. BCS EAST SQA group is definitely well disciplined. For software configuration control BCS uses the IBM ANMP (Account Network Management Programmer) data-base tool of which Livson was given an impressive demonstration. The weak point of ANMP compared to the RADC program support library is that it is based on the willingness of people to report to the configuration control management. During the demonstration the loudspeaker voiced a discussion about failed equipment due to an unreported software change...
Interestingly, the interdivisional or interfacility barriers in Boeing seem to be as high as within IAI. Not only was Boeing BCS EAST SQA unaware of software tools developed by Boeing Aerospace, but also by Boeing BCS West!
Art Russel made an interesting remark in explaining why his SQA group does not even strive to have independent testing capability: "The use of independent testing teams in Boeing has led to situations where the development group has shoved most or all testing down the throat of the independent testing group, but letting independent testing enter the picture only late in the integration with disastorous results".

7. INSTITUTE FOR COMPUTER SCIENCES & TECHNOLOGY
NATIONAL BUREAU OF STANDARDS
Washington D.C. 20234, May 22.
Software Tools Expert: Raymond C. Houghton, Jr.
Description: A demonstration of the NBS FORTRAN 77 analyzer on a VAX VMS 2.2 was given. The NBS analyzer is essentially a source instrumentation with somewhat extended capabilities to PROMO in our CDC C.A.D. library. The NBS analyzer is user friendly and interactive. It will be available almost for free through NTIS. A tool like this is included in software development work-benches like the GRC automated verification system and Colorado University TOOLPACK.
A demonstration of the University of California, Berkeley, virtual operating system utilities was given. It worked beautifully on the NBS VAX. This tool set has the following advantage over UNIX:

No changes in VAX operating system are needed. In our avionics real-time environment it is unlikely that we could give up VMS in favour of UNIX. The Berkeley system simply augments VMS with several dozens of utility programs. A switchover from the utilities back to VMS is instantaneous. Users can learn the new utilities in a gradual way without interference with their VMS activities.

The following documents were given to B. Livson by NBS personnel:

- NBS FORTRAN 77 Analyzer, Draft of User Manual by TRW Defence and Space Systems Group.
- The Software Tools Virtual Operating System Project, University of California, 16 December 1980.
- Draft of Tool Fair Proceedings, 5th International Conference on Software Engineering, 24 February 1981.
- Features of Software Development Tools, NBS Special Publication 500-74, February 1981.
- Computer Model Documentation Guide, NBS Special Publication 500-73, January 1981.
- Software Documentation for the Initiation Phase, FIPS Pub 64, 1 August 1979.
- Software Tools: A Building Block Approach, NBS Special Publication 500-14, August 1977.
- Technical Profile of 7 Data Element Dictionary/Directory Systems, NBS Special Publication 500-3, February 1977.
- NBS Software Documentation Standard, FIPS Pub 38, February 1976.
- Software Testing for Network Sciences, NBS Technical Note 874, July 1975.

Most of these publications and their in-process updates are in-machine readable form, too. B. Livson was tentatively promised a magnetic tape of all machine readable NBS documents in the fields of documentation standards, verification-validation & testing (VV & T), and software tools. If obtained, all IAI Engineering Division C.A.D. newsletter readers may route any of these documents to CDC lineprinters.

In addition to the above, NBS supplied B. Livson with its catalogue of software tools. The NBS software tools data-base incorporates most software tools catalogues, including TRW, Grumman, Boeing & Reifer catalogues. The in-process NBS tools catalogue will add SoHaR tools taxonomy descriptions, so that the NBS catalogue is just short of commercial recommendations.
As a final note, NBS also is active in software conversion methodologies and tools, the diversity of the U.S. Government computer facilities necessitating this.

8. DEPARTMENT OF COMPUTER SCIENCES, UNIVERSITY OF COLORADO

May 26.

Software Tools Experts: L.D. Fosdick & C.M. Drey.

Description: The first software reliability tool used by IAI Engineering Division was the Colorado University DAVE for global data flow anomalies analysis. Unlike a compiler that has to generate code on a program unit basis without knowledge of other program units until load/link phase, DAVE is capable of conducting data flow anomalies analysis across program unit boundaries. Most automated verification systems, including GRC FAVS, only have a limited global capability in that they can check common blocks and parameter lists in terms of lengths and types without entering into actual data flow analysis. Visits 3, 7 & 8 indicated that DAVE is still the best tool in its class.
L.D. Fosdick mentioned that the University of Minnesota has developed the MNF for CDC, a more powerful FORTRAN 77 compiler than our FTN5 compiler as far as diagnostics are concerned. The MNF incorporates some data flow analysis within a program unit. Our CDC Department may want to check what are the relative merits of MNF over our FTN. Tentatively, it seems that MNF is oriented towards development and FTN is optimized for production type compilation.
A new FORTRAN 77 compatible version, DAVE II will be ready by December 1981 for both CDC & VAX. DAVE II will permit most extensions to FORTRAN 77.
The second generation DAVE will still be a lot slower than a compiler, but not by an order of magnitude of old version. DAVE II will have improved recovery capability after phase 1 front end parsing.

All unacceptable source lines will be listed with an explanation (syntax error, unknown FORTRAN 77 extension, etc.) in contrast to the present version that stops at the first unacceptable source line.
An attempt will be made to use DAVE in a paperless environment.
The output file will be compressed to 80 columns.
B. Livson suggested the use of keywords for more convenient output file screening. He also suggested to output the program units call graph as a hierarchial block diagram for improved visibility.
The basic problem with DAVE, namely the large percentage of false alarms due to the commonplace bad programming practices will not be solved by DAVE II release.
The new TOOLPACK project will be available to the public domain only in late 1982. TOOLPACK will have similar capabilities as other automated verification systems.

9. INFORMATION SCIENCES INSTITUTE, UNIVERSITY OF SOUTHERN CALIFORNIA

Marina del Rey, May 28.
Contact: Bob Balzer.
Description: USC Inf. Sci. has a project to transform a formal software specification directly into executable code. If eventually successful, this would abolish the error prone and manpower consuming design and programming phases. The USC project includes:

- A formal specification language, GIST, and a tool directly transforming this into executable code.
- A tool aiding the stepwise refinement of natural language into GIST.
- A program transformations library. This is needed, since otherwise GIST would execute extremely slowly compared to present programming. Initially, a systems analyst has to pick up optimal program transformations so that the USC project includes a kind of very high level design phase.
- A software requirements testing tool allowing various mixes of dynamic and symbolic requirements testing.

Note that we can talk about dynamic requirements testing, since requirements are directly transformed into executable code. (Limited applicability.)

According to Daniel Berry from UCLA, it is unlikely that the Balzer-Prywes goal of direct execution of requirements will ever materialize. However, this "ultimate and impossible" goal may produce many interesting and useful side results.

10. SoHaR Inc.
1040 S. La Jolla Avenue, California, May 29.
Contact: Herbert & Myron Hecht
Description: SoHaR has the following three projects in-process:

A. Software Tools Taxonomy Project for NBS, see visit 7.
Taxonomy keys will be included in the NBS software tools data-base by the end of this year. SoHaR has subcontracted part of this job to D.J. Reifer from Software Management Consultants. According to H. Hecht, D.J. Reifer visited IAI in June.

B. Software error data collection project for RADC. The basic idea is program support library environment. A programmer cannot run his test cases independently, but must submit his runs via a librarian. This, according to H. Hecht, will ensure that software errors get reported properly. It is, however, easy to point out several flaws in this scheme, e.g. a programmer can claim that he needs to run the same test case with different probes or due to a change in software configuration. Thus, programmers can fairly easily obstruct software trouble reporting. In any case, this project does not seem to be practical in our environment. H. Hecht agreed that there is no way to justify a quantitative reliability goal for software so that software trouble reporting is just for the sake of a post mortem.

C. Fault tolerant software project for NASA experimental flight control system of a control configured vehicle. The idea is to build a software monitor to check some critical part of software. The monitor can fail in three ways:
- It is not executed at all.
- It erroneously activates a recovery module. This, in itself, should not be harmful provided that the recovery module is properly written.
- It fails to recognize a software error.

In summary, the approach of fault tolerant software development did not seem to be convincing. In addition to being time and manpower consuming, it may cause all kinds of integration problems. In the worst case, a fault tolerant software may increase the number of software errors. Besides, there may be such harmful psychological side effects as bad programming and testing of the baseline in the belief that a recovery is (always) possible.
This is similar to the danger of having an independent test team with no one doing any thorough testing. In B. Livson's opinion there is no choice but to insure that the development team does its job as well as possible. M. Hecht showed some of his work with software error tree analysis that seemed to be highly appealing. He will visit Israel 3-9 August 81, and he may lecture on the topic.

11. DEPARTMENT OF COMPUTER SCIENCE UCLA

1 June 81.

Contacts Daniel Berry and Rami Razouk

Description: Daniel Berry gave a demonstration of UNIX and the following documents:

UNIX Programmer Manual for

- Beginners
- Programmers
- Advanced Programmers
- Word Processing.

According to D. Berry, IAI would have to pay $40,000 for the first VAX and $15,000 for each consecutive VAX lincense to Bell Laboratories irrespective of the UNIX variate actually to be acquired.
Rami Razouk gave a lecture and demonstration of the UCLA tool, SARA, the System ARchitects Apprentice for requirement-driven, self-documenting design of modular, concurrent, hardware and software systems.
R. Razouk also gave the following documents:

- SARA Methodology
- Concurrent Software System Design by SARA (2 papers)
- Evaluation Methods in SARA - the graph model simulation
- The use of a Module Interconnection Language in SARA (2 papers).

SARA will be released on a commercial bases only after 5 years.

12. TRW DEFENCE & SPACE SYSTEMS GROUP

June 2.

One Space Park, Redondo Beach, California 90278

Contact: Richard L. Maitlen, Manager of Applied Systems Design Section (R. Maitlen is perhaps the No. 1 software tools expert in the U.S. software engineering industry, and started developing his first tools already in 1965).

Description: The visit to TRW had specific importance, since the TRW software development is quite similar to the IAI Engineering Division environment. TRW is the largest software developer for the U.S. Department of Defence, and has been responsible for huge projects such as Deck Boy for National Security Agency, Minuteman software, Over-the-Horizon Radar Project, Ballistic Missile Defence Advanced Technology Center, etc. TRW has had software development projects in excess of one million source lines in a single project. It was interesting to learn how TRW has systematically developed software tools packages to support this huge production environment: TRW almost solely develops its software in FORTRAN. It uses FORTRAN even in its commercial data processing application, where COBOL was traditionally used. TRW has augmented the capabilities of FORTRAN by various libraries optimized for bit manipulation, character processing and so on. The TRW argument is that the more modern languages such as JOVIAL 73 or PASCAL are not suitable for massive projects due to the deficiencies and partial lack of software tools. As an example, R. Maitlen mentioned the disastrous situation with the MX - missile software development, where TRW had the DoD directive to use JOVIAL 73. The J 73 compiler has serious errors and flaws. Besides, the set of software tools for J 73 is very limited. TRW even hesitates to use FORTRAN 77 due to compiler immaturity. In short, TRW is at least as conservative in the computer language aspect as we are. TRW computer network is based on CDC NOS, while TRW subdivision or department use a large variety of minis and midis as we do in the Engineering Division. TRW is at least as far from the paperless environment as we are; TRW has relatively few CRT's and almost no investment in graphics terminals, according to Mr. Maitlen. In fact, the only place working in a paperless environment at the time of B. Livson's visit was UCLA Computer Science Department.

The danger, according to him, is that too much time is spent to produce a proper requirement specification with very little time left for final testing and integration phases.

- In Maitlen's opinion the best design strategy would be a top-down skeleton program (in FORTRAN) with structured constructs and natural language comments, which is constantly refined and finally augmented by code. In his opinion the basic weakness of PDL (or our SDP) is that they get thrown away, since one cannot augment code into PDL. The nice thing in a real skeleton is that it can be compiled, statically tested, automatically flow-charted, and continued smoothly to coding.
  R. Maitlen gave to B. Livson the following two TRW documents:
  * TRW Software Tools Catalogue & Recommendations;
  * TRW Project Development using Software Support Tools, TRW Automated Software Tools Series, dd. September 1980.

In summary, IAI Engineering Division has comparable software development technology to TRW, but TRW's strength is the wide-spread utilization of software support tools.
As a final note, TRW does not market its tools, but is willing to develop tools on a contractual bases tailored to customer needs.
The price tag of such contracts is, of course, well over $ 100,000, whereas one can acquire ready tools for $ 10,000 - $ 50,000, and in some cases even for free (tools from nonprofit institutions such as universities; it is pointed out that often university tools are the best ones in being "intelligent". Most TRW tools are of highly mechanical nature).

The TRW recommendation for project development using software support tools is as follows:

A. Project Support Tools category consists of the Caine, Farber and Gordon Inc. Program Design Language (PDL) tool including graphic display capability, N^2 input-process-output interfaces charter tool, and word processor with all documentation being stored in permanent computer files. There is nothing unique in this category. In fact, IAI Engineering Division has a more advanced tool than PDL, the SDP, and an improved version of N^2 charts. What is relevant, however, is that TRW utilizes its project support tools systematically in every project.

B. Project Management Tools for scheduling, displaying budget data, identifying critical tasks etc. We are using PERT type tools as well as budgeting tools so that there is not much new for us in this TRW category. Neither TRW nor we (for that matter) have a truly interactive graphic tool, where the user could input with an "electronic pen" on the screen. The issue really is not what project management tools exactly do, but how user friendly they are. Such tools can be successful if, and only if causing minimal possible user resistance.

C. Software Tools for the Programming Environment.

This category is divided into two:

C1: Structured FORTRAN with precompilers from regular FORTRAN to structured FORTRAN to ease maintenance of old software projects, and from structured FORTRAN into regular FORTRAN enabling a more readable and maintainable software development. This issue has specific importance for such FORTRAN-LANDS as TRW or IAI Engineering Div. Included in this category is automatic listing indentation for increased readability, and Routine Level Control (RLC). The RLC assigns date and time for the last change in a routine, and assigns a checksum number. The checksum number ensures that the user will execute programs compiled to the last updated version number. In a mass production environment like TRW, this is very important.

C2: The Software Support Tools are divided into:

C2.1: Development tools. These include (semi) automatic common block insertion tools improving global variable management, which is very important in increasing the reliability and maintainability of large FORTRAN programs. We have overlooked this issue.

C2.2: Diagnostic Tools. This category is fully covered by an automated verification system, such as the GRC FAVS, and to a large extent by our project tools, namely DAVE and PROMO. Our two main problems are: (1) These tools are not extensively used. Perhaps by combining a number of such tools in a package like GRC did, will improve the situation. Specific attention has to be paid to make such a package as user friendly as possible. (2) These tools are wasteful in execution time. This problem will be solved by ensuring that the tools operate and output only over routines with changes. This necessitates a history file type data-base. Moreover, it is possible to preprocess programs, and then use a preprocessed version that will execute faster with the various tools. TRW tools include standards audit and a very strong variable analysis. This will be a natural part of our future AVS. Of course, TRW has automated flow-charters as we do. With structured FORTRAN such flow-charts are much more useful. Finally, TRW has automated documentation tools, the question being who has the time to read or even glance over all documentation produced.

Richard Maitlen had the following overview of modern software engineering:

- Software requirements tools such as the ISDOS PSA or TRW SREM cause a massive overhead in qualified manpower, and do not address to the problem of software requirements testing except that they increase traceability, and, of course, result in a better requirement definition.

13. GENERAL RESEARCH CORPORATION, SANTA BARBARA DIVISION

June 3.

Contact: William R. DeHaan, Software Marketing Director

Description: The GRC Software Workshop (SW) incorporates into one package tools similar to the ones forming the TRW support for programming environment. Starting September 1981, SW will be fully interactive MEMU-driven software. Note that neither the Colorado University TOOLPACK to be released late 1982 nor the present TRW tools are interactive. The interactive implementation of the SW is being done in a highly intelligent way: the screen is filled according to user desired hierarchial/nesting/indentation level. The menu-driven interactive SW demonstration was impressive in allowing the user (B. Livson) to see exactly the desired amount of details. Batch type tools such as DAVE have the drawback in "drowning" the user in paper. The second major advantage in the new SW is that it continuously updates the source and the associated data-base so that nothing is lost between two consequtive SW invocations. This feature, incidentally, was one of the main reasons why Colorado University launched its TOOLPACK project.

The GRC Software Workshop has the following tools:

1: RXVP80th (the commercial name for the RADC funded FORTRAN Automated Verification System). It consists of one compulsory component, namely the control and input component that among other things creates data-base interfacing with all other optional components. Presently, there are three optional components:

1.1 Static Analysis. This component does not have a truly global accross the program unit capability for global data flow anomalies analysis like our DAVE has. Possibly by the end of 1981 SW will have this global capability.

1.2 Execution Coverage Analysis. This tool is one of the best in its kind. It enumerates decision-to-decision paths and outputs the (accumulated) coverage for the various test cases only for these DD-paths and not for each and every statement. Our PROMO is essentially weaker than this tool. Also, the TRW instrumentation tools including the NBS FORTRAN 77 Analyzer seem to have a lesser capability.

3/6222-D

1.3 Document Generator

2: Preprocessors

2.1 V-IFTRAN structured FORTRAN verification tool has the following capabilities:

- It provides a full range of structures:
  - IF...OR IF...ELSE...END IF
  - WHILE...END WHILE
  - CASE OF...CASE...CASE ELSE...END CASE
  - DO...END DO (standard DO without label)
  - REPEAT...UNTIL
  - LOOP...EXIT IF...END LOOP
  - FOR...END FOR
  - BLOCK...END BLOCK & INVOKE

These structures provide GO TO- free and statement number - free replacements for the traditional GO TO, arithmetic IF, assigned GO TO, logical IF, computed GO TO and DO statements. Unlike FORTRAN 77, IFTRAN provides a truly block structured programming environment like the ALGOL-PASCAL family languages. It may be argued that some of the IFTRAN structures are needless, however. Note that IFTRAN can be used in KEYWORD mode in a way similar to the program design language (PDL).

V-IFTRAN has a debug printing capability of automatically printing I/Ø or intermediate variables by name and value. In addition, V-IFTRAN allows convenient insertion of assertions about initial statements, intermediate point conditions and final statements. If set OFF, the corresponding statements will be automatically changed to normal FORTRAN comment statements. It may be argued that a FORTRAN debugger has the same capabilities. V-IFTRAN, however, is powerful in enabling error handling blocks to handle with false assertions. In this respect, V-IFTRAN resembles ADA language.

2.2 Macro Facility for "repetitious coding/changing it later". The macro facility is similar to the VAX FORTRAN INCLUDE statement, and can be used for instance to modify common block copying, so important in TRW type programming environment.

2.3 REFTRAN global variable tool yielding the program units, common blocks and statements referencing a program variable or FORTRAN keyword. One of the best global cross-referencing tools.

2.4 FORTRAN STRUCTURER translates and indentates ugly FORTRAN into block structured IFTRAN. This tool will be made commercial by the end of 1981. RCA Corp. signed a special contract to structure 600,000 lines of ugly FORTRAN into IFTRAN indicating that this tool must have the efficiency of mass production tool.

GRW SW has been sold to TRW, General Electric, Hughes and various DoD facilities, and for a West German concern.

Boeing has pursued an interesting path: It acquired through RADC the semi-public batch ancestor of RXVP80, namely FAVS developed by Dr. Miller, a former GRC employee in the early 70's. Although one could acquire FAVS through RADC/U.S. General Services Administration nearly for free, its modernization would require several man years. RXVP80 source size is some 40,000 IFTRAN statements.

GRC provided B. Livson price lists and product summaries of their software workshop. The total price is $ 50,000-$ 55,000 with RXVP80 costing $34,000. The total price includes one year maintenance, a training course in Israel (2 days) and the associated installation on one mainframe, and documentation. It is stressed GRC provides the source. It would be possible to obtain the total package with a $ 35,000-$40,000 price tag in binary form, with no possibility to independently maintain the package or to add our own components or tools. GRC, in principle, is prepared to sell its package in numerous ways. In B. Livson's opinion, the best thing is either to acquire the whole package or nothing. The only other somewhat viable option would be to acquire 2 preprocessors, and package our present tools such as DAVE & PROMO. This, however, seems to be too little and too late. Besides, such an option would not be interactive. Neither would there be a joint data-base management.

14. LAWRENCE BERKELEY LABORATORIES, DEPT. OF COMPUTER SCIENCE

June 4-5.

Contact: Debbie Scherrer, Advanced Systems Research Group.

Description: Berkeley Laboratories have developed a portable "virtual operating system" with more than 100 software tools, both utilities and programming support tools, see Appendix. The Berkeley system has capabilities comparable to the BELL Laboratories UNIX 7.

The great advantage of the Berkeley system is that it is built on top of the VAX VMS operating system allowing users instanteneously to switch back and forth between VMS and the Berkeley system. Several hundreds facilities around the world adopted the Berkeley system.

This system could make our avionics and flight control system software development much more efficient. The basic idea is to allow people to concentrate on their software application without having to worry about utilities. As in the UNIX, an integral part of the Berkeley system is the standard input, output and error output interface that makes life so much easier for programmers. In addition, there is a method of pipelining or stacking utilities so that the output from one tool becomes the input of another tool. Although the Berkeley system was originally implemented for VAX VMS, it is not difficult to transfer it to our CDC CYBER NOS. Most system dependent quantities were pushed down into primitive function calls, which we have to implement. This will take a few man months, unless we will get a list of CDC CYBER primitives from the University of Arizona.

B. Livson was provided with full documentation, both programmer and installer manuals, and two magnetic tapes (one with VAX VMS implemented primitives, and a general purpose tape without primitives for CDC implementation). Once we get this implemented for our avionics VAX, the Berkeley system could be installed on any of the several IAI VAX machines, and with some systems programming effort on any IAI machine with at least 32 bit word length.

The Berkeley software tools are constantly being enhanced, and we were added to the list of more than 1,500 user groups receiving Berkeley Software Tools Communications. The immense success of both the Berkeley system and UNIX is partly due to SHELL, a powerful command line interpreter, that incorporates standard I/O, filters &pipes, multitasking,

flags, recursion and script files. SHELL can furthermore be used as a command language. SHELL is shortly described in Appendix. Hughes Aircraft Company has extensively used the Berkeley system, while Boeing Computer Services has experience with UNIX.

15. HUGHES AIRCRAFT COMPANY, RADAR SYSTEMS GROUP

June 9.

Contact: David Martin, Display Systems Dept., Man-Machine Simulation Section.

Description: David Martin is active in manually translating parts of the Berkeley software tools package into ADA. Hughes is evaluating U.S.A.F. ADA contracts proposals. The contract for validated ADA compiler, data-base manager, debugger and editor should be completed by the end of 1983. In a parallel development, Intel will release an ADA compiler for VAX ($ 30,000), and a development kit for the new 32-bit Intel i APX-432 micromain-frame. The total cost of an integrated VAX ADA compiler and i APX-432 development kit is around $ 80,000. Intel should release a military quality version of i APX-432 by early 1983. The commercial version including the ADA environment is already available. APX-432 has a sufficient band width for five parallel twin-processors. The slave processor monitors its master in a twin-processor. In case of a mismatch a twin-processor is shut-off leaving four out of five twin processors operational, so that it is possible to reach a high hardware reliability. In terms of performance, i APK-432 will not be much "faster" than i 8086 with a C-compiler. Hughes is evaluating i APX 432. It is unlikely, however, that Hughes will develop any major real time ADA software before 1984. Almost all Hughes software is written in COBOL and FORTRAN, the latter more recently in Ratfor after installation of the Berkeley system. Some of the more advanced Hughes projects have started to use Bell C-language with two options available to implement a UNIX-type software development workbench:

1: UNIX level 7 operating system.

2: Stanford Research Institute "EUNICE" that incorporates UNIX and/or Berkeley software tools and the C-compiler on top of VAX VMS.

Eunice costs only $ 2,500, but in both options one has to obtain a UNIX license from BELL Laboratories. For the first VAX machine this license costs $ 40,000 and for each consecutive VAX machine $15,000. This was one of the reasons why Hughes so far does not have a UNIX license but decided to install the Berkeley system. An interesting development system, the ZEUS, is available for ZILOG 8000. Zeus includes a C-compiler and most UNIX tools. Zeus allows time-sharing for 4-8 users. Zeus costs $ 30,000 including UNIX license fees. Zeus should be seriously considered for flight control system software development. From the point of software reliability, the C-language is quite problematic. C-compilers carry no strong-type checking such as array boundaries or parameter list checks. C-compilers like FORTRAN compilers carry out no global analysis. The draw-back with C compared to FORTRAN is that the set of software tools for C is very limited. The advantages of C over FORTRAN, of course, are huge: readability, excellent access at machine level, powerful high level data types including pointers, fast and compact object, etc.

Software development in Hughes is totally dispersed with no organization or budgeting for software tools or utilities. David Martin is working on the software tools area in a "partisan way" without authorization or budgets. The attitude in Hughes is to allow individuals to conduct such activities to the extent that this benefits (or does not interfere) with project activities. Each of the many projects has its mini or midi computer (VAX, PDP 11-series, SEL etc.) with software and hardware engineers working closely together with projects being independent of one another. The operating environments are usually very small, 5-8 users to a midi or mini.

Hughes has not acquired or developed any methodologies or tools for software requirements analysis according to D. Martin's knowledge.

16. BOEING COMPUTER SERVICES (BCS) COMPANY
SPACE & MILITARY APPLICATIONS DIVISION (SAMA)

Seattle, June 10,

Agenda:

16.1 BCS/SAMA Overview by Harold E. Olson

16.2 BITS by Roger A. Franklin

Description: BCS developed Boeing Intelligent Terminal System combines the best of both worlds: centralized and distributed data processing. The idea is that each user has limited local processing, memory and storage capability, and is also integrated into a central data processing network. Several hundred BITS are operational, the eventual requirement calling for 10,000 BITS at a cost of $150m. The relatively high unit cost, $15,000 for each multipurpose terminal is mainly due to the requirement for graphics capability.
Document: From mini to Micro - The Intelligent Terminal.

16.3 SEWS: A Software Engineering Workstation
by Gary Campen

Description: SEWS is a huge relational data-base incorporating most software development: Requirements & Design, Data Dictionary, Source Code & Documentation, Configuration Control, and Project Management Data. SEWS integrates the following tools:

- Dump/Load Utilities
- Query/Report System
- Specification Language Translator
- Analysis & Reporting Tools
- Interactive Graphics
- Code Generation (COBOL from detailed design) & Test
- Document Generation

SEWS development will take a number of years, the customer being NASA Langley Research Center. SEWS tries to solve the present problem of primitive data-base architecture: Network model with Codasyl Chained Records, or hierarchial model with multiple text files.

Future architecture has:

- User interface (unified command language)
- Tools
- Data-base interface (data language)
- A relational data-base with inverted files.

In addition to user friendliness (unified command language with an integrated tools package), BCS claims that tools like PSA can be run 10-100 times faster. SEWS does not add to the analytical power of present day tools such as PSA, but provides integration of data-bases, and an interactive user-friendly environment.

Document: SEWS/G. Kampen 11/13/80/

16.4 Integrated Verification & Testing System (IVTS) by Robert Hite

Description: IVTS provides verification and testing analysis tools for HAL/S programs. HAL/S is a PL/I like NASA language for space shuttle flight control software. BCS will deliver IVTS by February 1982. IVTS seems to be less advanced than Colorado TOOLPACK, GRC software workshop or TRW toolset. IVTS covers static analysis (incorporates DAVE), dynamic analysis (assertions & execution statistics gathering), and an experimental tool for symbolic execution. Leon Osterweil and his Colorado associates have done most consulting work to develop tools for symbolic execution and data flow analysis of concurrent processing anomalies. B. Livson is in the process of checking the feasibility of the new Colorado material. IVTS does not really integrate its tools. IVTS command language is interactive, but the tools are batch.

Document: IVTS/NASA Summary.

Note: Most of the documents given to B. Livson are still at the drafting stage.

16.5 IPAD by L. Stein, R. Johastone, L. Corthims

Description: IPAD is a massive PASCAL software package (140,000 lines with more than 300 man years of development) for Boeing/NASA C.A.D., C.A.M, and inventory control. Readers interested in IPAD may obtain IPAD user manual introduction from B. Livson. Boeing had to develop this huge package without DEBUGGER, and seem to have paid a very high price in choosing PASCAL.

As a coincidence, the same Minnesota University PASCAL compiler is being used by both BCS and IAI Engineering Division for our CDC machines.

16.6 ARGUS by Bonnie McIntosh

Description: ARGUS is a software development workbench with the following components:

- DYNA: Dynamic analyzer roughly comparable to NBS analyzer or IAI/Raytheon PROMO.
- STRADA: An interactive graphics data-flow modelling system with an integrated data dictionary subsystem. STRADA is being developed to support the Yourdon-Constantine structured analysis and design methodology. STRADA is mainly a design tool with the capability of generating PSL/PSA definitions of the data-flow model directly from the data-flow diagrams themselves.
- Architecture Analyzer: Includes results of analysis of code, data and process descriptions, access assertions, interfaces, etc. in a data-base. It is not clear what kind of analysis is performed. This tool is years from implementation.
- Project Management Aids: Available on stand-alone microprocessor systems (BITS). These aids are quite primitive: Project milestone charts, organization charts, and hardware configuration diagrams.
- FORMS Mode: Full-screen forms for data entry in a data-base. No analysis of data-base.

In summary, both STRADA and architecture analyzer are years from implementation. ARGUS does not have a command language interpreter glueing its tools together like UNIX or Berkeley packages. ARGUS does not seem to be a well-defined system.

Document: Concepts & Prototypes of ARGUS by Leon G. Stucki and Harry Walker, Software Engineering Environments, North Holland 1981 preprint.

16.7 Software Reliability Investigations by J.R. Brown and J. Skrivan.

Description: On-going project for NASA to test software developed by two independent teams from a common specification. J.R. Brown has studied how to assign probability distributions by dividing input space into subsets. These investigations do bot seem to be practical. Large companies like BCS nevertheless finance quantitative software modelling on a moderate bases, 2-3 man year budgets.
Documents: BCS/NASA Investigation Summary; Testing for Software Reliability by J.R. Brown & M. Lipow (TRW Software Series).

16.8 Rel-Time Software Checkout by R.L. Glass

Description: Up-to-date technology is used to specify, to design and to code software. However, software is debugged using archaic technology:

- Octal, hexadecimal output
- Machine code patching
- No printers, no pre-planned debug print, no memory for debug package.
- Instruction level simulators function at machine code level.

Boeing's answer to real-time software checkout is remove source code errors in the host computer environment, and defer target computer checkout until late integration and system test phases. Program unit testing and most of software integration testing should be done in the host computer environment. The combined module-integration test phases consume approximately 75% of the checkout resources. Boeing emphasizes:

- Host environment simulator
- All high level language version of code
- Host machines with very large or virtual memory.
- Host compiler producing both host and target machine object code.
- Instruction lever simulator to simulate target machine hardware on the host.

Host can use instruction level simulator to analyze sizing, accuracy, support software, perhaps timing problems. Boeing does not suggest any elimination of target computer checkout. Problems, such as the hardware behaving differently from the environment simulator, must be target computer checked.

Boeing conducted an extensive error analysis of two large real-time projects, MPRT and E-3A AWACS. Some 124 errors were studied with the following findings:

- Environment simulator could have detected 90-out-of-124 errors.
- Data tracing 41/124.
- Test coverage analyzer 40/124.
- Assertions 37/124.
- Dumps (snap or post-mortem) 35/124.
- Eyeball checks (structured walk-throughs, peer code reviews - call it whatever you like) 90/124.
- Sneak analysis 20/124.

Most of these categories are strongly overlapping. Neverthless, three conclusions can be drawn:

1: Eyeball checks are highly effective, although not sufficient to check safety critical software. Some 70-75% of software errors could be removed by eyeball checks.

2: Environment simulator is the only methodology with comparable efficiency to eyeball checking.

3: Software tool packages like GRC software packages, Colorado TOOL-PACK, TRW tool set, Boeing IVTS etc. could discover about 50% of all errors (inferences from the above data are quite difficult).

It is likely that all of the above means (1-3) are necessary to check both flight control and navigation&weapon delivery software. Software errors accounted for more than a third of all system problems in Boeing E-3A AWACS and MPRT project. It is likely that this percentage will grow. The above conclusions are summarized in D180-25859-1 Boeing reports (2) obtainable from B. Livson.

The following is a summary of R.L. Glass' recommendation for a minimum standard software tool-set based on the Minimum ADA Programming Support Environment (MAPSE):

- Requirements Definition: No recommendation
- Design : PDL
- Compiler/Assembler
- Linker/Loader
- Conditional Compilation
- Global Cross-Reference List
- Call Graph Generator
- Timing/Performance Analyzer
- Configuration Management Tools
- Word Processing
- Data-base/File Manager
- Text Editor (both line-editor and screen editor)
- File Comparator.

IAI Engineering Division has all of the above tools excluding automated configuration management. We have some CAD tools and CDC utilities that enable limited configuration management.

17. HIGHER ORDER SOFTWARE, Inc.

June 11.

Contact: Jack Rosenbaum.

Description: A tool called "FAME" generating a relational data-base from HOS input is being marketed. HOS methodology has been enhanced by abstract data types, and loop structures. An interface between HOS data-base and PSA is currently being developed. HOS Inc. markets "FAME" at $ 15,000 for perpetual license, and prices annual maintenance and updates at $ 7,500. By the end of 1981 HOS Inc. will release a larger tool, "USE-IT", that generates PL/I code from detailed design HOS. It is questionable as to whether such code generation is practical. IAI R & D had extensive contacts with HOS Inc. concerning these tools. One of the reasons, why IAI R & D did not acquire these tools, was HOS Inc. insistence on supplying only binary object, and not source code. Rosenberg's remark that customers are guaranteed of source kept in a closed bank vault in case of HOS going out of business did not make a good impression on B. Livson.

APPENDIX: BERKELEY SOFTWARE TOOLS

1. TOOLS

ar ..... archive file maintainer
cat ..... concatenate and print text files
ch ..... change text patterns
comm ..... print lines common to two files
cpress ..... compress input files
crt ..... copy files to terminal
crypt ..... encrypt and decrypt standard input
date ..... print the date and time
dc ..... desk calculator
detab ..... convert tabs to spaces
diff ..... isolate differences between files
echo ..... echo command line arguments
ed ..... editor
edln ..... in-core editor
entab ..... convert spaces to tabs and spaces
expand ..... uncompress input files
fb ..... search blocks of lines for text patterns
field ..... manipulate fields of data
find ..... search a file for text patterns
format ..... format text
includ ..... file inclusion preprocessor
kwic ..... prepare lines for keyword-in-context index
lam ..... laminate files
ll ..... print line lengths
macro ..... general-purpose macro processor
mcol ..... multicolumn formatting
mv ..... move (rename) a file
os ..... convert backspaces into multiple lines
pl ..... print specified lines/pages in a file
pr ..... print file
ratfor ..... Ratfor preprocessor
rev ..... reverse lines
rm ..... remove (delete) files
roff ..... [see "format"]
sedit ..... stream editor
sh ..... command line interpreter
show ..... show all characters in a file
sort ..... sort and/or merge text files
spell ..... locate spelling errors
split ..... split file into pieces
tail ..... print last lines of a file
tee ..... copy input to output and named files
tr ..... character transliteration
tsort ..... topologically sort symbols
uniq ..... strip adjacent repeated lines from a file
unrot ..... unrotate lines prepared by kwic
wc ..... count lines, words, and characters in files
xref ..... make a cross reference of symbols

2. SUBROUTINES AND PRIMITIVES

(* indicates that the implementation of the routine is system-dependent
# indicates that the routine may, in some cases, be system-dependent)

definitions ..... standard Ratfor definitions

File Manipulation
#amove ..... move (rename) file1 to file2
*close ..... close (detach) a file
*create .... create a new file (or overwrite an existing one)
*getfyp ..... get type of file (character or binary)
*isatty ..... determine if file is a teletype/CRT device
#mkuniq ..... generate unique file name
*open ... open an existing file for reading, writing, or both
*remove ..... remove a file from the file system

I/O
fcopy ..... copy file in to file out
*flush ..... flush output buffer for file 'fd'
getc ..... read character from standard input
*getch ..... read character from file
#getlin ..... get next line from file
*nofe ..... determine current file position
#prompt ..... prompt user for input
putc ..... write character to standard output
*putch ..... write character to file
putdec ..... write integer n in field width >=w
putint ..... write integer n onto file fd in field width >=w
#putlin ..... output a line onto a given file
putstr ..... write str onto file fd in field width >=w
*readf ..... read from an opened file
*remark ..... print single-line message
*seek ..... move read/write pointer
*writef ..... write to an opened file

Process Control
*spawn ..... execute subtask

String Manipulation
addset ..... put c in array(j) if it fits, increment j
addstr ..... add string s to str(j) if it fits, increment j
clower ..... fold c to lower case
concat ..... concatenate 2 strings together
cpoc ..... copy string-to-string
ctoi ..... convert string at in(i) to integer, increment i
ctomn ..... translate ascii control character to mnemonic
cupper ..... convert character to upper case
equal ..... compare str1 to str2; return YES if equal
esc .... map array(i) into escaped character, if appropriate
fold ..... convert string to lower case
getol ..... generalized character-to-integer conversion
getwrd . get non-blank word from in(i) into out, increment i

gitoc ..... generalized integer-to-character conversion
index ..... find character c in string str
itoc ..... convert integer to character string
length ..... compute length of string
lower ..... convert string to lower case
mntoc ..... ascii mneumonic to character
scopy ..... copy string at from(i) to to(j)
sdrop ..... drop characters from a string
skipbl ..... skip blanks and tabs at str(i)
stake ..... take characters from a string
stcopy ..... copy string at from(i) to to(j); increment j
strcmp ..... compare 2 strings
strim ..... trim trailing blanks and tabs from a string
substr ..... take a substring from a string
type ..... determine type of character
upper ..... convert string to upper case

Pattern Matching

amatch ..... look for pattern matching regular expression
getpat ..... encode regular expression for pattern matching
makpat ..... encode regular expression for pattern matching
match ..... match pattern anywhere on line

Command Line Handling

*delarg ..... delete command line argument number "n"
*getarg ..... get command line arguments
gfnarg ..... get next filename argument
query ..... print command usage information

Dynamic Storage Allocation

dsfree ..... free a block of dynamic storage
dsget ..... obtain a block of dynamic storage
dsinit ..... initialize dynamic storage

Symbol Table Manipulation

delete ..... remove a symbol from symbol table
enter ..... place symbol in symbol table
lookup ..... get string associated with name from hash table
mktabl ..... make a symbol table
rmtabl ..... remove a symbol table
sctabl ..... scan all symbols in a symbol table

Date Manipulation

fmtdat ..... convert date to character string
*getnow ..... get current date and time
wkday ..... get day-of-week corresponding to month-day-year

Error Handling

cant ..... print "name: can't open" and terminate execution
error ..... print single-line message and terminate execution

Miscellaneous

*endst . close all open files and terminate program execution
*inifst .. initialize all standard files and common variables

3. sh - shell (command line interpreter)

SYNOPSIS

sh [-vnxc] [name [args]

DESCRIPTION

Sh is a command line interpreter: it reads lines typed by the user and interprets them as requests to execute other programs.

Commands.

In simplest form, a command line consists of the command name followed by arguments to the command, all separated by spaces:

command arg1 arg2 ... argn

The shell splits up the command name and the arguments into separate strings. Then a file with name command is sought; command may be a path name to specify any file in the system. If command is found, it is brought into memory and executed. The arguments collected by the shell are accessible to the command. When the command is finished, the shell resumes its own execution and indicates its readiness to accept another command by typing a prompt character.

If file command cannot be found in the current directory or through its pathname, the shell searches a specific system directory of commands intended to be available to sh users in general.

An example of a simple command is:

sort list

which would look for the tool "sort" in the current directory, then in the system directory, and then sort the contents of file "list", printing the output at the user's terminal.

Some characters on the command line have special meanings to the shell (these are discussed below). The character "@" may be included anywhere in the command line to cause the following character to lose any special meaning it may have to the shell (to be "escaped"). Sequences of characters enclosed in double (") or single (') quotes are also taken literally.

Standard I/O

Shell programs in general have open three standard files: "input", "output", and "error output". All three are assigned to the user's terminal unless redirected by the special arguments "<", ">", "?", ">>", "??", (and sometimes "-").

An argument of the form "<name" causes the file "name" to be used as the standard input file of the associated command.

An argument of the form ">name" causes file "name" to be used as the standard output.

An argument of the form "?name" causes the file "name" to be used as the standard error output.

Arguments of the form ">>name" or "??name" cause program output to be appended to "name" for standard output or error output respectively. If "name" does not exist, it will be created.

Most tools have the capability to read their input from a series of files. In this case, the list of files overrides reading from standard input. However, many of the tools allow the user to read from both a list of files and from input by specifying the filename "-" for standard input. For example,

```
roff file1 - file2
```

would read its input from "file1", then from the standard input, then from "file2".

Filters and Pipes.

The output from one command may be directed to the input of another. A sequence of commands separated by vertical bars (|) or carets ("^") causes the shell to arrange that the standard output of each command be delivered to the standard input of the next command in sequence. Thus in the command line:

```
sort list | uniq | crt
```

"sort" sorts the contents of file "list"; its output is passed to "unic", which strips out duplicate lines. The output from "uniq" is then input to "crt", which prepares the lines for viewing on the user's crt terminal.

The vertical bar is called a "pipe". Programs such as "sort", "uniq", and "crt", which copy standard input to standard output (making some changes along the way) are called "filters".

Command separators

Commands need not be on different lines; instead they may be separated by semicolons:

```
ar t file; ed
```

The above command will first list the contents of the archived file "file", then enter the editor.

The shell also allows commands to be grouped together with parentheses, where the group can then be used as a filter. For example,

```
(date; cat chocolate) | comm vanilla
```

writes first the date and then the file "chocolate" to standard output, which is then read as input by "comm". This tool compares the results with existing file "vanilla" to see which lines the two files have in common.

Multitasking

On many systems the shell also allows processes to be executed

in the background. If a command is followed by "&", the shell will not wait for the command to finish before prompting again; instead, it is ready immediately to accept a new command. For instance,

```
ratfor ambrose >george &
```

preprocesses the file "ambrose", putting the output on "george". No matter how long the compilation takes, the shell returns immediately. The identification number of the process running that command is printed. This identification may be used to wait for the completion of the command or to terminate it.

The "&" may be used several times in a line. Parentheses and pipes are also allowed (within the same background process).

Script files.

The shell itself is a command, and may be called recursively, either implicitly or explicitly. This is primarily useful for executing files containing lines of shell commands. For instance, suppose you had a file named "nbrccunt" which looked like this:

```
echo "Counting strings of digits"
tr <program 0-9 9 | tr !9 | wc -c
```

These commands count all the digit strings in "program". You could have the shell execute the commands by typing:

```
sh nbrcount
```

The shell will also execute script files implicitly. For example, giving the command

```
nbrcount
```

would cause the shell to notice that the file "nbrcount" contained text rather than executable code. The shell would then execute itself again, using "nbrcount" as its input.

Arguments may also be passed to script files. In script files, character sequences of the form "$n", where n is a digit between 1 and 9, are replaced by the nth argument to the invocation of the shell. For instance, suppose the file "private" contained the following commands:

```
cat $1 $2 $3 | crypt key >$4
ar u loveletters $4
```

Then, executing the command

```
private Dan John Harold fair
```

would merge the files "Dan", "John", and "Harold", encrypt them, and store them away in an archive under the name "fair".

Script files may be used as filters in pipelines just like regular commands.

Script files sometimes require in-line data to be available to them. A special input redirection notation "<<" is used to achieve this effect. For example, the editor normally takes its commands from the standard input. However, within a shell procedure commands could be embedded this way:

```
ed file <<!
editing requests
!
```

The lines between <<! and ! are called a "here" document; they are read by the shell and made available as the standard input. The character "!" is arbitrary, the document being terminated by a line that consists of whatever character followed the <<.

Shell flags.
The shell accepts several special arguments when it is invoked. The argument -v asks the shell to print each line of a script file as it is read as input. For instance,

```
sh -v private Jasmine Irma Jennifer twostars
```

would print each line of the script file "private" as soon as it is read by the shell.

The argument -x is similar to the -v above except that commands are printed right before they are executed. These commands will be printed in the actual format the system expects when attempting to execute the program.

The argument -n suppresses execution of the command entirely.

The argument -c causes the remaining arguments to be executed as a shell command.

Termination.
The shell may be left by typing an end-of-file or by typing "logout" as a command.

FILES

Scratch files for pipelines are created with the names "Pn" where n is the number of the pipe on the command line (e.g. 1, 2, etc.). Scratch files named "doc" are created for "here documents".

SEE ALSO

The Unix command "sh"
The Bell system Technical Journal, vol. 57, no. 6, part 2, July-Aug 1978

DIAGNOSTICS

cannot locate shell
Issued whenever the shell is attempting to spawn a copy of itself to process a script file. The shell was unable to find the correct file. (The file is set in a string declaration in the "scrf" routine.)

cannot spawn background process