AD-A052 996 TRW DEFENSE AND SPACE SYSTEMS GROUP REDONDO BEACH CALIF F/G 9/2
NSW FEASIBILITY STUDY.(U)
FEB 78
F30602-76-C-0270
UNCLASSIFIED RADC-TR-78-23 NL

1 OF 2
ADA
052996

RADC-TR-78-23
Final Technical Report
February 1978

NSW FEASIBILITY STUDY

TRW Systems & Space Group

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nationals.

RADC-TR-78-23 has been reviewed and is approved for publication.

APPROVED: Richard A. Robinson
RICHARD A. ROBINSON
Project Engineer

APPROVED: Wendall C. Baumann
WENDALL C. BAUMANN, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss
JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISCP) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

(18) (19) **REPORT DOCUMENTATION PAGE**

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-78-23

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE *(and Subtitle)*: (6) NSW FEASIBILITY STUDY

5. TYPE OF REPORT & PERIOD COVERED: (9) Final Technical Report.

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

8. CONTRACT OR GRANT NUMBER(s): (15) F30602-76-C-0270

9. PERFORMING ORGANIZATION NAME AND ADDRESS
TRW Systems & Space Group
One Space Park
Redondo Beach CA 90278

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
P.E. 6372F
J.O. 55500845

11. CONTROLLING OFFICE NAME AND ADDRESS
Rome Air Development Center (ISCP)
Griffiss AFB NY 13441

(11) Feb[illegible]78

(12) 98 p.

13. NUMBER OF PAGES
92

14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*
Same

(16) 5550

(17) 08

15. SECURITY CLASS. *(of this report)*
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

Same

DDC RECEIVED APR 21 1978 F

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Richard A. Robinson (ISCP)

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*
Software Systems
Software Engineering
Computer Networks

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

This report describes work performed to determine the feasibility of using the National Software Works (NSW) to support weapon systems software developments. It contains a discussion of the issues involved in using computer networking resources in support of such projects, and has some suggestions for applying initial NSW capabilities to prototype testbed projects. Appendices contain evaluations of some potential NSW tools, including the JOVIAL Automated Verification System (JAVS) and the On-Line System (NLS), and suggests a configuration management plan for NSW tools.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

409 637

TABLE OF CONTENTS

APPENDIXES

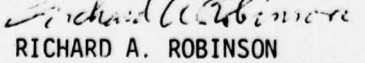
ILLUSTRATIONS

EVALUATION

The purpose of this study was to determine the feasibility of using the National Software Works (NSW) to support the development of large scale weapon systems software and to assist the government in the development of NSW capabilities appropriate to that support. This report describes the work performed and the conclusions formed as a result of that work.

The contractor was provided access to computers at several ARPANET host sites, acquired a Display NLS Work Station, and used ARPANET and NSW resources in support of a candidate weapon system software project. Extensive use was made of the NLS System to generate engineering reports and project documentation, as well as JOVIAL and FORTRAN Computer Programs. The writing and review of documents, including periodic reports and configuration and quality management procedures, was greatly simplified using on-line resources. Other tools destined for the NSW were also evaluated for possible application. Telecommunication features of the ARPANET/NSW, in particular, were evaluated and found to be very useful in the creation and coordination of requirements documentation. Utilization of network resources would also have made it possible to postpone the delivery of the OTH-B project computer for several months at a substantial savings in labor, investment and operating costs. Finally, in a mature NSW, a full complement of OTH-B specific tools could be assumed to exist, along with many others of use to the project, and would not have required development or installation into NSW.

As a result of this effort, a feasible determination has been made that the NSW is a viable resource that can be used to support the orderly development of major weapon systems. It is accordingly planned to perform an applications experiment using ARPANET/NSW resources to support major software system developments. Effort to be conducted fits into the RADC Technology Plan (TPO 5/R5A) and will make use of tools that have been specifically installed into the NSW for application on the OTH-B Project. Scope of the experiment remains to be determined, particularly as it relates to how environmental simulation and other debugging processes which are deemed necessary to test the interfaces of an embedded computer system will be conducted. Further consideration will also be given to how extensive an environment is required to satisfactorily demonstrate the usefulness of the NSW over the full "life cycle" of a project.



RICHARD A. ROBINSON
Interactive Processing Section
Computer Technology Branch

1.0 INTRODUCTION 1

The objective of this effort is to conduct a study to determine the feasibility of using the NSW (National Software Works) to support the development of large scale weapon systems software, and to assist the government in the development of NSW capabilities appropriate to that support. 1a

This report describes the work performed and the conclusions formed as a result of that work. 1b

As part of an effort to understand the environment of the NSW, TRW became active on the ARPANET, acquired a Display NLS terminal, and used various other terminals such as Tl Silent 700, Execuport, Logabax, Tectronix and DTC/300 at port speeds of 300 and 1200 bps. The NLS system was used to generate engineering reports as well as JOVIAL and FORTRAN computer programs. 1c

Other tools destined for the NSW, notably the JOVIAL Automated Verification System (APPENDIX B), were studied, as were the line oriented editors XED, TECO and SOS (APPENDIX D). 1d

In order to assure a realistic approach to tool application, TRW searched among its ongoing weapon system software projects and chose the Over the Horizon Radar Project (OTH-B) as a representative testbed in which a fair variety of software production tools are in use, and a project which has a strong, innovative approach toward software tools and methodology. A 300 page volume of the OTH-B Design Specification document (C-5) was transfered from the UNIVAC 1110 Word Processing system to NLS in order to study the benefits of the NLS structure in the modification, study and implementation of design documents. 1e

Finally, the NSW itself was exercised in its preliminary form. 1f

From this familiarity with the environment in which the NSW was conceived, and its intrinsic familiarity with the weapon system software environment, TRW was able to recommend a set of policies, requirements and procedures by which a demonstration of NSW should be conducted, in order that data taken from the demonstration have maximum meaning (Chapter 6.0). 1g

These principles are general enough to be useful in evaluating the few tools now available on the NSW which are of value to weapon system software projects and, hopefully, those tools which will be added to the NSW in the future. In order to gain the generality required of this dual role, the experiment design section is more extensive and at a higher level than would have been required for the evaluation of a few specific tools. 1h

Because the NSW does not now have a formal plan for managing the tools within its inventory, and because such management will become necessary in an operational environment involving weapon system projects, a draft proposal for such a plan was developed, and is included as Appendix E of this report. 1i

2.0 THE NATURE OF A WEAPON SYSTEM SOFTWARE PROJECT 2

Weapon system software projects share several characteristics pertinent to NSW: 2a

1. They are conducted according to government purchasing requirements for configuration and quality control. 2a1

2. They involve computers which interface with (or are embedded in) other devices or systems. 2a2

3. They involve complex algorithms or processes. 2a3

While not all weapon system software projects display all of these characteristics to the same extent, a knowledge of them is helpful in understanding the role of tools in the production of this kind of software. 2b

The government purchasing requirements are embodied in a bewildering array of documents which supercede, supplement, or explain each other. One of the problems of a weapon system software project is to understand the particular set of standards under which it is to perform. Having understood the requirements, the project management must disseminate the information to the performing team. Fortunately or otherwise, the standards as presented in the documents are usually modified by agreement of the government and the project. 2c

At TRW, much of the dissemination of performance requirements information necessitated by the government purchasing requirements is performed by instruction pertaining to the Unit Development Folder (UDF), which is the worker's outline and repository of all of the elements of a unit of software. Since the form of the UDF is similar from project to project, the detail differences can be absorbed by the worker, who need not study the source control documents and agreements. 2d

Another source of control information at TRW with which the individual worker must be familiar is the Project Software Standards and Procedures Manual, which contains design and implementation formats, conventions and instructions for documentation and computer code. 2e

The progress of a project is managed according to the establishment of a series of baselines. A typical set of events and associated baselines is presented in figure (1). The primary documents of interest in this example are listed on the left side of the figure. The "A spec" is a document presenting the government's needs, the "B spec" presents the functional requirements of a system that would meet the government's needs, the "C spec" is a design, including a program listing, which satisfies the functional requirements, and the "test spec" is a plan by which the project can demonstrate that the product satisfies the functional requirements. Besides these primary documents (and the Project Software Standards and Procedures

Manual), various hardware and software interface control documents are normally produced, along with a Users Manual. 2f

A typical cost allocation is presented in figure (2). It should be noted that the costs allocated do not include the gathering of requirements or the maintenance of the software in its field use. These phases are highly variable by nature, and sometimes exceed the total of the allocated costs. 2g

The computers for which weapon system software is coded range from the on-board, special purpose, highly integrated control computers used in aircraft and ICBM navigation and guidance, to ground based general purpose computers which are only loosely integrated with the environment which they describe or control. Applications, such as some ICBM targeting programs, may interface only through conventional magnetic tape files and may, therefore, be suitable for ordinary batch or timeshare execution. Even these applications, however, involve design considerations which imply intimate consideration of complex data interfaces within the object system. 2h

The complexity of weapon system algorithms or processes is in contrast to other fields of software production such as business data management or communications switching in which the fundamental processes are simple, even though the resultant program is often complex due to the sophistication required by the large volumes of data which must be processed. 2i

These three characteristics of weapon system software have largely determined the kinds of tools which have been developed for use by the projects. 2j

The government purchasing requirements give rise to a variety of report generation tools which aid in tracing of requirements through the various specification documents. Software interface documents are produced through the aid of tools which summarize the interface information presented by the individual component designers. Formal testing is facilitated by the application of tools which describe code standards adherence, code branches traversed during formal test, data referenced but not set during test, etc. 2k

The physical environment interface characteristic of weapon system software projects gives rise to a variety of tools which simulate aircraft, rocket and other environments. Because the embedded computers are often too small or special purpose to support any kind of tool (even an assembler) many projects maintain tool libraries on large, general purpose computers for the support of embedded computers. Finally, the embedded computers themselves are often simulated by tools resident on general purpose computers. 2l

The complexity of the algorithms and processes which characterize weapon system software gives rise to the need for diagnostic tools which analyze and trace the logical flow of programs and components. Tools of this type may be used to find unexpected branches, data necessary to cause execution of branches and related statistics. They may also find the portions of time

consumed by various parts of large programs, and thus aid in optimizing complex processes.

2m

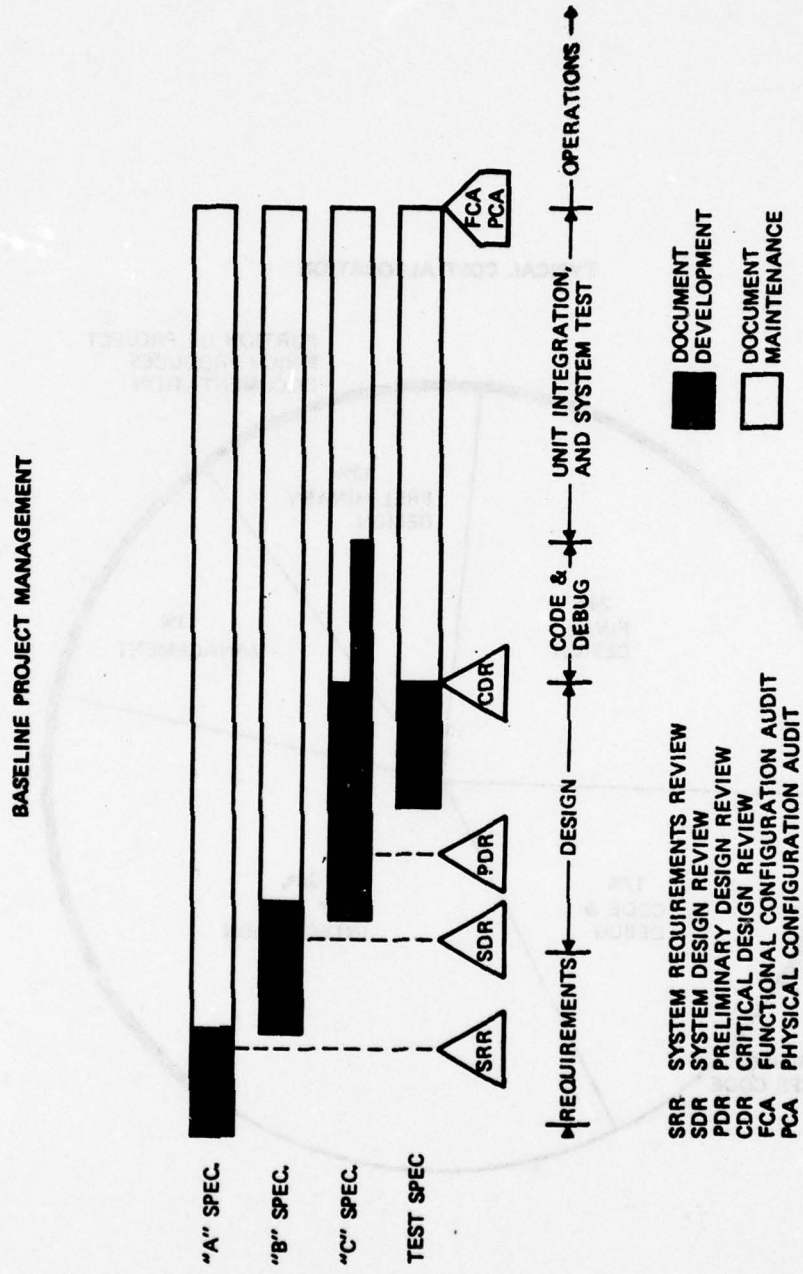


Figure (1). Baseline Project Management.

2n

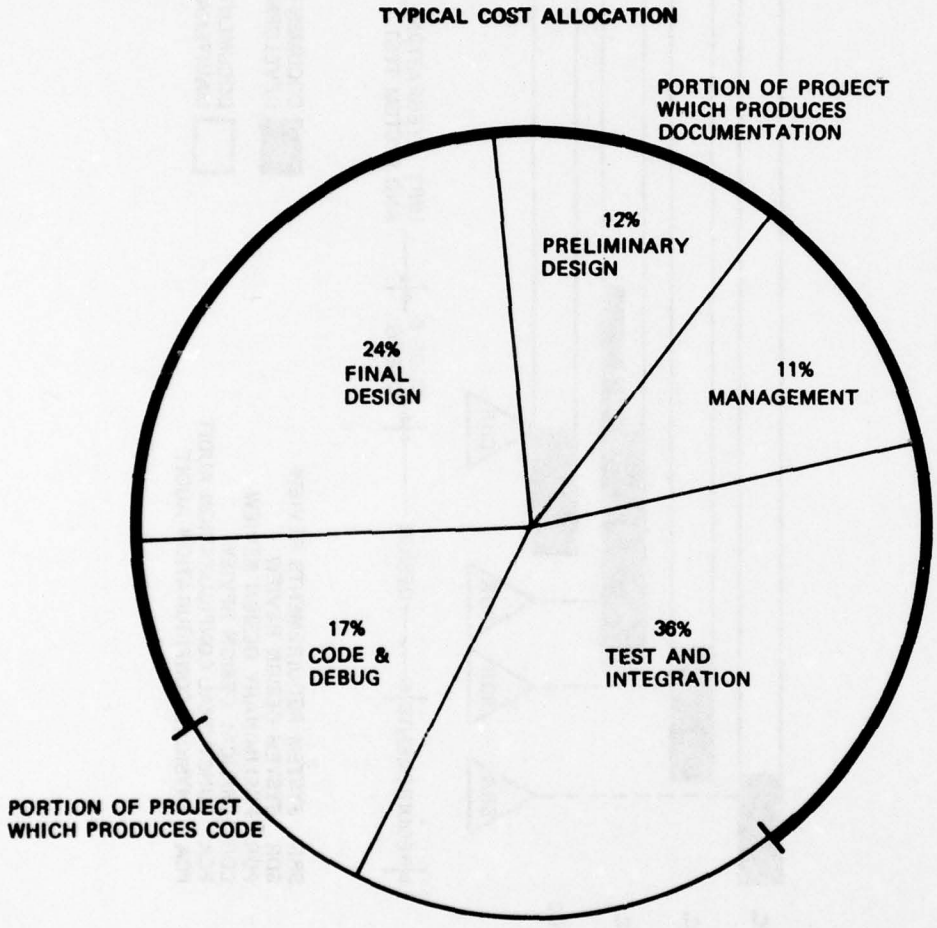


Figure (2). Typical Cost Allocation.

3.0 TOOLS PRESENTLY USED IN WEAPON SYSTEM SOFTWARE PRODUCTION 3

The tools inventory at TRW comprises over 300 programs, including special purpose, obsolete and general engineering tools. For the purposes of describing a generally applicable set of tools, a few of those programs have been chosen, and classified according to use by software projects. Commercially available tools such as PDL would be included here, but compilers, operating systems, data base managers and other support tools normally supplied by the computer manufacturers would not. 3a

Brief descriptions of several representative programs are included in Appendix A. 3b

3.1 TOOL CATEGORIES 3c

1. Program Management Tools 3c1

Tools in this category are decision making aids which are applicable to proposal and program managers for planning and control purposes. They encompass cost/schedule control and reporting, PERT networks, work breakdown structures, performance measurement, cost estimating, design to cost, etc. Some typical computer program examples are: 3c2

SPREAD, WBS, TCOST, PRS, HARP III.

2. Engineering and Scientific Tools 3c3

Tools in this category are analytical aids which are applicable to computer scientists and systems engineers for evaluation purposes. They encompass system simulators, language processors, math models, computer hardware/software models, etc. some typical computer program examples are: 3c4

SALSIM, COMPSIM, NETSIM, SPRINT, DPAD, DPSS, SINDA, UFSS.

3. Design Tools 3c5

Tools in this category are decision-making aids which are applicable to project technical managers and functional managers to establish operating rules for software design, development and test. They encompass program architecture, language augmenters, programming standards, software converters, pretest code checkers, interface checkers, data base management, etc. Some typical computer program examples are: 3c6

TREE-META, PCP, DEVISE, COMGEN, STRUGN, SINGAN, PDL, SPRINT

4. Development Tools 3c7

Tools in this category are programming aids which are applicable to programmer/analysts to design, code, checkout, integrate and test computer

programs, components and data bases. They encompass logic/equation generators, document generators, report generators, comparators, monitors, editors, flowchart generators, diagnostic, etc. some typical computer program examples are: 3c8

JIGSAW, BLKGEN/SPECPN, DPNDCY, AUTODOC, TIDY, TATTLE, FLOWGEN, SCS, DOCGEN, MPS, NODAL, DOCET, CODE AUDITOR, COMGEN, IJST, STRUCT, TEXTEDIT, CADMUS.

Of particular interest in this group is the program JIGSAW, which combines the functions of a structured JOVIAL J3 to compilable JOVIAL translator, a code auditor and branch instrumenter in one preprocessor. 3c9

5. Test Tools 3c10

Tools in this category are design support aids which are applicable to developers, independent test conductors and product assurance inspectors for development, acceptance, demonstration and operational testing of coded products. They encompass standards enforcers, test case generators, test data generators, test predictors, retest generators, consistency checkers, etc. Some typical computer program examples are: 3c11

ATC, CAD, PACE, ANODE, RETEST, COMPAR, UCA, NODAL, AUTORETEST, DRIVER.

6. Verification, Validation and Certification Tools 3c12

Tools in this category are reliability tools which are applicable to product assurance engineers to assure the software was developed in accordance with the applicable specification(s) and satisfactorily performs the functions for which it was designed. Some typical computer program examples are: 3c13

ATC, CAD, PACE, ANODE, RETEST, COMPAR, UCA, NODAL, AUTORETEST, DRIVER.

7. Configuration Management Tools 3c14

Tools in this category are management control tools which are applicable to program configuration management specialists for the identification, control and status accounting of computer programs, components and changes thereto. They encompass program identification, baseline control, document control, tape control, library control, interface control, data base control, etc. Some typical computer program examples are: 3c15

CPAL, CHECKSUM, CONFIG, FORMAN, GIM/GIM II.

3.2 OTH-B EXPERIENCE 3d

The Over the Horizon Radar System (OTH-B) is an ESD contract to the Military Systems Department of the General Electric Company in Syracuse, N.

Y. The software is on subcontract to TRW Systems and Energy in Redondo Beach, California. The system, consisting of High Frequency radio antenna arrays, data compactor computers, and a UNIVAC 1110 computer for central display and control, is to be installed in Maine. The UNIVAC computer is installed at TRW for the software production phase of the contract. 3d1

The OTH-B project at TRW began using software production tools shortly after the beginning of its contract, some two years ago. The first tool used was the General Purpose Simulation System (GPSS) which was available to the project on an IBM 370. The GPSS study effort was directed to establishing rough estimates of computer resource utilization by the various software components. The study results indicated a disk access contention problem which dictated modifications to top level design. The GPSS model continues in use, with increasingly refined data. 3d2

The next tool to be used was a text editor. Since the UNIVAC 1110 had not been delivered at the time of document preparation for Preliminary Design Review (PDR), the TRW Time Share System (TSS) was used. The TSS editor MPS (Manuscript Preparation System) was used by secretaries to input hand-written design documents. Later, the MPS files were transferred to the UNIVAC and modified for use by the editor available on that machine. The documents for Critical Design Review were produced by updating the PDR documents. Secretaries and programmers entered text at the eight CRT terminals available on the machine. 3d3

Before PDR, the data interfaces and the control interfaces between software components were under the management of tools especially created for the project. These tools print "calls" and "called by" lists as well as "set by" and "used by" lists for all components and routines, based on design data. The JOVIAL COMPOOL input file is produced by this system, as well as many of the design document tables and lists. 3d4

Because OTH-B uses a structured programming language, a preprocessor was built to convert the project standard "constructs" to J3 statements. The preprocessor also functions as a code auditor (warns the programmer or test conductor of standards violations) and optionally instruments the J3 statements for static and dynamic structure and flow analysis. 3d5

The project and its customers also considered the use of the tool PDL (Program Design Language) as a replacement for detailed flow charts. Reluctance to give up flow charts and lack of a UNIVAC version of PDL dictated against PDL for OTH-B. 3d6

Documents and code are currently maintained on-line by the programmers and secretaries, who manage project files using the normal facilities of the EXEC-8 operating system. Programmers create code using the editor, submit new or modified files to the preprocessor, thence to the JOVIAL compiler, and finally to the project test bed for checkout. As more and more checked out components are added to the test bed, previously functioning components

will be seen to fail, sometimes necessitating reversion to an earlier version of the test bed, which is usually available due to the configuration control procedures in effect. 3d7

In considering the application of NSW to the OTH-B project, several advantages can be seen to accrue: 3d8

1. Delivery of the UNIVAC 1110 could have been delayed for several months, at substantial savings in labor, investment and operating costs. During the interim, other computers in the NSW could have supplied the relatively modest early computing requirements of the project. If an 1110 had been available within the NSW, the delay could have been even longer.

2. The telecommunication feature of the NSW would have been particularly useful in the creation of the software requirements documents. Partly because of the geographical distance between the contractors, technical and management knowledge sometimes was difficult to share. Joint use of text files would have saved much of the calendar time which was expended on technical documents requiring collaboration.

3. In a mature NSW, the equivalent of the tools mentioned above could be expected to reside (along with many others of use to the project), and would not have required development by the project.

4. NLS would appear to be a much more effective system with which to have generated the project documentation. With NLS, the principal documents might have been created by the programmers in an on-line environment instead of off-line, to be copied on-line by secretaries. Ease of generation and maintenance would have encouraged the on-line writing and review of ancillary documents, including periodic reports and configuration and quality management documents.

4.0 TOOLS CURRENTLY AVAILABLE IN THE NSW 4

The NSW, in its current experimental state, has only a very few tools available for use. As of November 9, 1977, the tools available were: 4a

TECO, SOS, XED, QEDX, PRIM, UYK20, U1050, EMLOAD, MAC20, MACRO20, CMS2M, PLM, ASM80, FORTRAN, SPPCOBOL. 4a1

The first four programs are editors, the next seven are concerned with the PRIM computer emulation capability at ISI or with special purpose tools allied to that capability and used chiefly by the Navy at NOSC. PLM and ASM80 are cross compilers for the Intel 8080. The FORTRAN is the IBM version at UCLA, and SPPCOBOL is a structured preprocessor for COBOL, although no COBOL compiler exists in the NSW. 4b

The encapsulation efforts at MULTICS will soon result in the availability of a JOVIAL compiler, as well as JAVS, the JOVIAL Automated Verification System. Also to be available soon is the NLS documentation system and an interactive debugging system for use with the higher order language CML. 4c

The TRW program JIGSAW is soon to be available to this project in two versions, one on the MULTICS and the other on a TENEX machine. This availability will enable a demonstration of the NSW capabilities on the OTH project at TRW, or any other project using a structured JOVIAL (J3) Higher Order Language. 4d

The editors are excellent line oriented systems which are somewhat typical of the large number of editors available from computer manufacturers and other sources. 4e

NLS, being a document structure oriented system, differs from the others and constitutes a unique capability, not currently available to any weapon system software project. The use of the CRT display and cursor input system associated with NLS constitutes another facet of the NLS system which is of potential use to weapon system software projects. 4f

The NLS system does not totally obviate the need for line oriented editors because many printing devices are line oriented, and because NLS may be less cost effective in some (or many) applications. Thus, because there will always be line oriented files and line oriented tasks, it would seem that the line oriented editors will always play a prominent role in the NSW. 4g

5.0 NSW AS A SUPPORT SYSTEM 5

If the NSW had all of the tools so far mentioned, would it be a viable support for weapon system software projects? The answer would have to be yes, qualified only by the assumption that the system, including the hosts, affords fast enough response that the user is not greatly hindered while performing his tasks. Indeed, large portions of some projects could be performed with the editors, compilers and test tools available right now, or scheduled in the near future. 5a

It seems likely that any project using NSW in the foreseeable future would wish to add its own tools to the complement, however. These new tools might be project specific or, hopefully, new and of use to others. The ease with which either class of tool may be added to the NSW will be a powerful factor in the relative success of NSW. 5b

The very nature of the software industry dictates that its tools must continually change. After all, only one program of a given sort need be produced. The next program (or the next project) can be expected to be different enough that some new tool may need to be developed, or an old tool modified. Meanwhile, other projects must rely on the fact that the tools they are using do not change. Some modifications to old tools would be invisible to old users, but other changes could render a tool useless to particular projects. 5c

The need for continual change in the tool inventory, together with the dependency of a project on the integrity of its tools, points out the need for a strong tools management system within the NSW. 5d

For a project to commit itself to the use of tools over which it has no control would require that the project be given assurance that an organization of highest technical and management reputation can and will guarantee the integrity of the tools and the system. To a slightly lesser extent, the project must be assured that adequate computer resource would be available and that consultation in the use of (and diagnosing of apparent malfunction of) tools would be available. 5e

Should a project not receive such assurances, it will invariably elect to do without tools or, increasingly, to build its own tools, on its own machine. The well known "Not Invented Here" syndrome may cause some projects to elect to build their own tools anyway. 5f

Several options appear open to the government in providing the management control necessary for the NSW: 5g

1. The government could minimize the necessity for control by placing the configuration and maintenance responsibilities on the individual tool builders. While this might relieve the NSW of the necessity to control each tool, the responsibilities would have to be delegated in contract

form, and the user would be required to interface with a variety of contractors. 5g1

2. The government could appoint a single agency as the manager of the NSW tool inventory. That agency could have provision for user consultation and the power to set policies for the acceptance, maintenance, and dissemination of tools. Such an agency should not be responsible for the expenditure of funds for the acquisition of new tools, in order that impartiality be assured relative to the support of old tools. 5g2

3. The government could hire a private firm to manage the tool inventory. The firm would have approximately the same power as the previously mentioned government agency, except that its policies and actions would be subject to government review. The advantage of such a course might be that a firm already experienced in tools management could achieve a high level of efficiency sooner than an inexperienced agency. 5g3

In any event the overall problem of support guarantee must be addressed. Are government contractors to be held legally responsible if they are unable to fulfill commitments due to unexpected failure of the NSW or its tools? Contractors normally would have recourse relative to their sub-contractors in such an event. What recourse could they expect to have relative to the NSW? It would not seem that this problem has an immediate, universal solution. Perhaps the answer will come from the individual project contractors and sub program offices as they attempt to place covenants in contracts concerning government responsibility for the NSW. The problem would seem to be solvable, but not by any one authority now constituted. 5h

A candidate system for handling configuration control problems such as: 5i

- o procedures for accepting the tools and relevant documentation 5i1
- o controlling design changes 5i2
- o processing discrepancy reports 5i3
- o managing the overall control process 5i4

is presented in Appendix E of this document. 5j

Technically, the NSW specification would seem to be broad enough to promise, depending on implementation, that the user can pretend that he is using tools on a single machine, and that all appropriate files can be transported from tool to tool with no restrictions in format. While the implementation of this concept remains (in a practical sense) to be demonstrated, there would appear to be no insurmountable obstacles to achieving the goal. 5k

6.0 TO EXPERIMENTALLY DEMONSTRATE THE NSW 6

This section describes requirements for designing and implementing an experiment to evaluate NSW usefulness. Paragraphs that follow describe the requirements that should be satisfied in selecting a meaningful application of NSW capabilities, requirements on the experiment environment, and requirements to be satisfied by NSW users in reporting results. Also proposed in this section are an attempt to define the attribute "usefulness" and some considerations in analyzing and evaluating this attribute. 6a

6.1 INTRODUCTION 6b

The ideal environment for evaluating the use of NSW to improve human intellectual performance would include a large population of users, a well understood problem situation, and a short solution period. This set of conditions would enable evaluators to carry out reliable experiments for testing well-formulated hypotheses using a large sample size to account for human variability. 6b1

Controlled experiments are particularly difficult to set up in a real world software development environment. Due to variabilities in the development process itself, repeatability is prevented and quantitative results are almost always inconclusive. However, qualitative results from a meaningful application can be conclusive in establishing whether a tool or technique is useful in solving generic types or classes of problems. 6b2

The challenge is to do enough evaluative work to indicate convincingly the qualitative strengths and usefulness of NSW and to do it at a minimum cost. To accomplish this, an experiment must be conducted which is based on an application of NSW to a software development or maintenance project which is representative of real world weapon system software development. Sufficient qualitative (and quantitative) data must be collected as the experiment progresses to support analysis. 6b3

6.2 DEFINITION OF USEFULNESS 6c

Before proceeding, it will be necessary to define what is meant by the characteristic USEFULNESS. Borrowing from the reference *, which defines a characteristic called USABILITY, we can state that the NSW possesses the characteristic USEFULNESS to the extent that it is convenient and practicable to use. To be sure, quantifiable measures of convenience and practicability are lacking in this statement, but we can make some further statements relating to USEFULNESS which will aid in our qualitative assessments. 6c1

First we can hypothesize that application of NSW will not preclude performance of any software development or maintenance tasks to be performed. Evaluation of this hypothesis for each of a series of generic

* Reference 1: Characteristics of Software Quality, B. W. Boehm et. al., TRW-SS-73-09, December 1973.

tasks, covered in Section 6d3, will be binary in a sense. If application of NSW precludes performance of a task for some reason, USEFULNESS is zero. 6c2

If NSW usage aids in the performance of these tasks, then USEFULNESS is non-zero and we have to evaluate the extent to which it aids each task, i.e., is convenient and/or practicable. Here we can provide the evaluators with some questions to answer in establishing a qualitative figure of merit for USEFULNESS. 6c3

1) Do the NSW capabilities allow the user to perform a task he would not be able to perform without NSW?

2) If NSW capabilities duplicate automated capabilities currently available to the user, do NSW capabilities represent an improvement over these conventional capabilities?

3) Do NSW capabilities perform in a fashion which facilitates or encourages their use elsewhere in a different project environment (e.g., development of a logistics system)?

4) Is use of NSW capabilities in performance of a task supported by adequate user documentation? Is the user/NSW interface adequately human engineered?

5) Does usage of NSW capabilities interfere with the ancillary aspects of software development such as configuration management and user status reporting?

6.3 REQUIREMENTS ON THE NSW APPLICATION 6d

Selection of an environment in which to evaluate NSW capabilities should be viewed in the context of what is typically known as the software life-cycle. This life-cycle begins with basic research and feasibility studies performed prior to issuance of a request for proposal (RFP) by the acquisition agency. The next phase is proposal evaluation and selection of a contractor. This is followed by the familiar software development and maintenance cycle made up of the phases shown in figure (3). 6d1

SYSTEM REQUIREMENTS

SOFTWARE REQUIREMENTS

DESIGN

CODE & DEBUG

UNIT TEST

INTEGRATION TEST

SYSTEM TEST

OPERATIONS & MAINTENANCE

Figure (3). Software Development Cycle

Although each of these phases of the life cycle could benefit in observable ways from capabilities of the NSW, the most attractive object of an evaluative experiment appears to be the development (software requirements through maintenance) phases of the lifecycle. The reasons for this, in addition to the need to keep the scope of the feasibility study to a realistically achievable level, are as follows: 6d2

The software development and maintenance phases are most critical in determining total system costs. Therefore, the most cost effective benefits of a system such as NSW can be expected to be those which aid these phases of the lifecycle.

The modern programming practices currently being required of DoD software developers have created a more disciplined environment in which to set up experiments, collect data for evaluation, and analyze results. That is, data to be used as a basis for evaluation is more readily available and more easily collected in the development and maintenance phases.

Some secondary benefits of evaluating NSW capabilities in the development and maintenance phases are expected in the area of research and technology transfer. That is, certain tools, new techniques, and needs for communication will be suggested by the attempts to name the NSW capabilities and then to evaluate those capabilities.

The worth of any tool or technique to the weapons system software development process must be determined from a set of development tasks, products, and measures of success (or measures of accomplishment) characteristic of a "typical" software development project. Examples include the following: 6d3

Development Tasks-

Requirements specification, documentation, validation, traceability, and evaluation.

Preliminary design, based on specified requirements.

Detailed design, based on approved preliminary design and baselined requirements.

Coding and debug, based on the approved (and documented) detailed design.

Unit testing at the smallest unit of compilable code, for example, a subroutine in FORTRAN terminology.

Integration of software components and exercise of the functional capabilities with increasing data stress, functional stress, and "computer sciences" stress.

Formal system level or validation testing to software and/or hardware requirements.

Operational testing or demonstration of system capabilities in the operational environment.

System engineering of the computer software, hardware, data structure, data values, and user interfaces.

Configuration management of software and documentation, including configuration identification, configuration status accounting, and product control.

Contract administration and data management according to CDRL requirements.

Planning and control activities, including budget control, schedule (PERT or critical path techniques), and work breakdown structure considerations.

Products-

In a current software development environment governed by contractual standards (e.g., MIL-STD 483, 490, 1521, etc.), software standards for development and coding, (e.g., structured programming, commentary blocks, and formatted source code) and management standards (e.g., chief programmer teams, structured code and design inspections, unit development folders, etc.) the products of the development process are very well defined. Formats of the software documentation are formal, evidence of the success or failure of producing these products is typically a contract deliverable, and these contract deliverables are listed in the CDRL contained in the contract. Software is, with increasing frequency, being treated as something other than data, as was the previous practice. Typical documents required contractually, in MIL-STD 490 terminology, include the following:

1. System Specification (Type A Specification)

2. Development Specification (Type B5 Specification)

3. Product Specification (Type C-5 Specification)

4. Data Base Description Document

5. Interface Control Document

6. Users Manual

7. Test Specification

These specifications document progress of each phase of the development process and are the basis for formal configuration baselines established during the lifecycle of the software product.

Ancillary or support documentation not contractually required but produced as a by-product of the development process are also a product of the typical software development process. Examples of this type of documentation include the following:

1. Status Reports

2. Meeting Minutes

3. Action Item Records

4. Maintenance Plans

5. Configuration Management and Quality Assurance Plans

Measures of Success-

Each phase of the development cycle includes some point at which the progress of the project is reviewed and the product at that point is reviewed. The typical review points are the System Requirements Review (SRR), Preliminary Design Review (PDR), Critical Design Review (CDR), and Physical and Functional Audits of the product (FCA and PCA). Other review points may be directed depending upon the contract, the software development contractor, and the specific requests of the software procuring agency. The key issue concerning each of these review points in the development process is that the chance exists to reach agreement that the progress on the product is adequate to proceed to the next phase of development. The documentation and demonstrable performance of the software product form a baseline, as defined in MIL-STD 483, against which subsequent development is to be based.

6.4 REQUIREMENTS ON THE EXPERIMENTAL ENVIRONMENT 6e

In order to assure that the feasibility study is representative of a modern development environment, it will be necessary to impose some requirements on the experiment. 6e1

1) There must be a software product developed or maintained. For our purposes implementation of previously built software in a new environment constitutes a development activity.

2) There must be requirements for the software being implemented, developed, or maintained. These must include functional, performance, and interface requirements. It is necessary that these requirements be available prior to the beginning of the experiment.

3) The experiment must be conducted in a "project" environment. There must be a project manager responsible for the tasks and products (documentation and software). There must be a customer, preferably acting as a System Project Office (SPO), which participates in technical reviews, gives technical direction, and requires status/progress reporting at regular intervals.

4) There must be a project review authority which monitors the experiment as a nonparticipant.

5) There must be some tests performed on the end product to demonstrate that the requirements stated at experiment start were actually satisfied.

6) There must be an individual who operates as the experiment data recorder, member of the configuration control board (CCB), and keeps the configuration management records.

6.5 TRAINING REQUIREMENTS FOR EXPERIMENT PARTICIPANTS 6f

Participants in the experiment must have training and practice in interfacing with NSW prior to the beginning of the experiment. It is estimated that a minimum period of from one to two weeks be allotted for the purpose of instruction by NSW trainers, familiarization with NSW documentation, and practice sessions on the terminal. 6f1

A longer period would be desirable if the effects of human variability in learning were to be minimized. However, the short learning period is more representative of the real world of software development where training often takes place "on the job". And, since the objectives of the experiment are more geared to assessment of usefulness, valuable lessons may be learned about the user interface and adequacy of user documentation if people not intimately familiar with NSW are used. 6f2

There are several goals that should be met in selecting participants, however. Participants should be accustomed to interacting with an on-line system, preferably with terminals similar to those which give access to NSW and ARPANET. Experiment results should not be influenced by unfamiliarity with terminals and interactive systems. Participants should also be current in development techniques employing modern programming practices

and use of software tools. Individuals with approximately one year of weapon system software development experience using online terminals should be a goal in staffing the experiment. 6f3

6.6 DATA COLLECTION DURING THE EXPERIMENT 6g

Two generic types of data should be collected during the experiment: performance data and usage data. Both can be used in the evaluation process. Data must be supplied by each participant involved with NSW usage, including the manager of the evaluation project. These two data types are described in the paragraphs below along with suggestions for instructing the provider of data (i.e., the NSW user and his manager) in the use of data collection forms and the importance of both qualitative and quantitative data. 6g1

PERFORMANCE DATA

This is data related to the actual performance of software development activities using the NSW. As participants exercise the capabilities of the NSW they will be required to keep a log of information, both positive and negative, that can be used in qualitative evaluation. This will include such things as:

- o problems/positive aspects of NSW capabilities
- o problems/positive aspects of tools made available through the NSW
- o comments on NSW documentation
- o comments concerning the user interface, including the terminal
- o suggestions for new or augmented capability

Log entries must be correlated with specific development activities such as document preparation, code debugging, measuring test thoroughness, etc. The specific NSW capability being exercised must also be identified for each entry. A final requirement is that entries must be made on the day the activity was performed in order to capitalize on the freshness of the experience in the mind of the performer.

Prior to beginning the experiment, users must be instructed in the objectives of the experiment, i.e., to support a qualitative assessment of usability and to suggest possible improvements. Experience in collecting subjective data from large software development projects has shown that the suppliers of data are able to

provide more useful material if they know what the objectives of the experiment are.

Clearly, the users will be affected by the knowledge of experiment objectives and the need to write information down as it becomes available.

Effects of the foreknowledge are unavoidable in a search for subjective data. The effects of writing things down can be minimized if the user is encouraged to make his log entries after logging off.

A sample log is presented here:

```
NSW USER'S LOG
ENTRY NO.
DATE:
USER NAME:
LOGON TIME:
NSW CONFIG:
[] PROBLEM
[] COMMENT
[] RECOMMENDATION/IMPROVEMENT
USER ACTIVITY:  (Space for written reply)
NSW CAPABILITIES USED:  (Space for written reply)
ENTRY (PROBLEM/COMMENT):  (Space for written reply)
ALTERNATIVE OR WORKAROUND TAKEN:  (Space for written reply)
```

USAGE DATA

Usage data is more easily quantified and may be collected automatically to some extent. The principle measures of usage will be in manhours (or CPU units) required to accomplish specific NSW tasks (e.g., operate NSW to transfer a file, transfer source code into an environment, etc.) and manhours to accomplish more generic software development tasks (e.g., debug a subroutine of a given size, prepare a document of a certain type and size, report status, etc.).

Automated data collection may necessitate augmentation of NSW usage reporting capabilities but will benefit the experiment by reducing the interference of data collection activities with NSW user activities. Capabilities for automated collection should satisfy the following requirements:

- Wall clock time at specific NSW actions (the time between commands may suffice)
- CPU time segregated by machine and identified as NSW time, tool time, overhead, idle time, etc.

- The number of error returns from each tool session

The automated data collection function should collect all of the data indicated by the above stated requirements, but should not be restricted to that data. The goal should be to collect all independent data available, subject to reasonable restraints on storage and processing resources used.

6.7 ASSESSMENT OF USEFULNESS 6h

Prior to beginning the experiment it will be necessary to identify a project review board that will ultimately assess the USEFULNESS of NSW in the experiment. Members of the review board should have backgrounds in software system development consistent with the experiment that is selected. Project management, system engineering, and integration and test backgrounds should be represented, as well as first level technical supervision. 6h1

It is important that the project review board, including the customer (SPO) and project manager, define the tasks to be performed and agree upon completion criteria for each task. This will enable production of schedules for task start and end dates, and the determination of task interdependencies. From this network of scheduled tasks a series of evaluation points can be selected. 6h2

At each of the evaluation points, when task completion criteria are met, those involved with assessing USEFULNESS will draw on available data and interviews with project performers to answer questions given in section 6C and other questions which will become obvious from task selection. It is important that evaluation be performed at these scheduled points, when collected data and support information is conveniently available. 6h3

For each task that is completed using NSW capabilities USEFULNESS will be nonzero. For tasks that were aided by NSW capabilities, USEFULNESS will be positive; for those impeded by NSW capabilities, USEFULNESS will be negative. 6h4

USEFULNESS RATINGS

4	-	high positive
3	-	moderate positive
2	-	low positive
1	-	slight positive
0	-	of no use
-1	-	slight negative
-2	-	low negative
-3	-	moderate negative
-4	-	high negative

Ratings will be the "subjective" product of the evaluation committee for each NSW capability used to complete a task, since no controlled experiment is planned nor is there an adequate benchmark data base available to make comparative evaluation possible. Numeric ratings are provided for purposes of assessing relative worth. 6h5

It is expected that one of the positive outputs of the assessment process will be realization of a set of expanded capabilities that would enhance NSW usefulness in future applications. These expanded capabilities must be recorded for each task for use in determining potential impact on conventional development techniques. 6h6

6.8 ASSESSMENT OF IMPACT 6i

A picture of NSW capabilities on the software development process should be indicated by the experiment, if tasks selected are representative of tasks currently performed in weapon systems development. 6i1

Impact assessment will be another task of the project review authority. Here again the assessment will be subjective and a function of the tasks selected and the backgrounds of members of the review authority. Principle questions that must be addressed and substantiated, where possible, by experimental data: 6i2

Are NSW capabilities applicable to weapon system software developments which vary in size, management structure, contract type, governing contract requirements, operational environments, geographical displacement of project performers and resources, etc.?

Are NSW capabilities applicable to developments other than weapon system software?

Are NSW capabilities degraded by such factors as number of users, generally available resources, support services such as maintenance, etc.?

Does usage data obtained during the experiment indicate that training and support documentation was acceptable and sufficient?

Does the NSW system provide disciplined and timely update and maintenance services for users?

The objective impact assessment will be to determine if NSW capabilities, both existing and projected, further the state of the art of weapon system software development; and if these capabilities are generally applicable to all projects or singularly applicable to a limited class of development types. 6i3

6.9 ASSUMPTION 6j

Having described the approach to evaluation and the generic objectives of experiments to determine usefulness, it might be wise to briefly state one underlying assumption which has bearing on what the experiment will not cover. The experiment will not be a test of NSW capabilities, per se. If failures are encountered, they will be noted in the data collection process. However, it is assumed that NSW capabilities will have been validated and shown to be performing as intended prior to the beginning of the experiment. 6j1

Secondly, the experiment will not be a test of tools made available by NSW. There will be evidence of tool usefulness, and problems traceable to the tools will be documented; however, the tools are assumed to have been previously validated. 6j2

7.0 SUMMARY

7

In comparing the relatively few tools in the NSW to the many that just one weapon system software contractor (TRW) uses in its project work, one might conclude that NSW must have an equal number to be of any significant value. On the contrary, most of the work on any given project can be done with a small number of tools. Surely the tools must begin with a compiler and perhaps, depending on the circumstances, an assembler. Beyond that, any tools added would be more than many projects now use. 7a

Assuming the presence of the approved compiler and the editors, including NLS, the NSW could be demonstrated. Compared to projects which use no other tools, the NSW would probably be a viable place in which to build software; but the integration and test of that software would be very difficult, if not impossible, because of the lack of environment simulation and other debugging tools necessary to test the interfaces of an embedded computer system. 7b

Therefore, in order to consider demonstration of the NSW over the full life cycle of a project, a tool which enables simulation and test should be added to the NSW. Such a tool is the Software Design and Verification System (SDVS) recently installed in the Avionics Laboratory at Wright-Patterson AFB. With the addition of the capabilities found in this tool, a large category of avionics software could be produced and delivered online. 7c

Also of significant value for the future of NSW would be an attempt to transfer the TRW Unit Development Folder (UDF) technology solidly into the online environment, perhaps through an NLS subsystem. With such a subsystem, programmers would be aided in ensuring that all phases of development are completed; including design, code, checkout, integration, documentation, design review, etc. 7d

By maintaining scheduling information online with the UDF, managers would be able to tap the pulse of the project, receiving up-to-date estimates of progress and identification of trouble spots. By having the design and code online and collected together, it becomes possible to incorporate requirements definition technology, such as the TRW SREP/RSL methodology; design technology, such as the TRW DEVISE system, the Air Force CARA system, and the Caine-Farber-Gordon PDL system. It will also be possible to do enhanced verification of requirements traceability; we may find that specific HOL extensions to enable requirements tracing are desirable, and that we can develop automated traceability verification tools. The long run tie-in to automatic program proof is also a possibility. 7e

As a first step, it is recommended that a feasibility analysis be performed which would define the initial online adaptation of the UDF, and would define the requirements and preliminary design of a prototype UDF subsystem. The next effort would be to construct and demonstrate the prototype subsystem in the context of NSW. 7f

In further summary of the role which NSW can play in support of weapon system software projects, the requirement for project "telecommunications" should be mentioned. 7g

Because relatively large amounts of information can be transported by the NSW among the often separated elements of typical projects, many hours, days, and, in effect, months can be saved by using the familiar terminals and network to share information. The nature of the NSW is far beyond that of the teletype network that it resembles, in that the NSW will allow the users to work on shared files, and to receive information and transmit it via a familiar, convenient instrument that is useful for many other project activities. 7h

The implications of network telecommunications suggest that, when the procuring agency, the prime contractor and appropriate subcontractors are all attached to the NSW, agreements written in network messages and available to all entitled parties will replace slow traveling conventionally mailed memos and unreliable recollection of telephone conversations. When one considers that documents (often the subject of the memos) can be controlled, modified, accessed and shared by the entitled parties using NSW, the potential for improvements in efficiency of production and quality of documentation would seem to be enormous. 7i

Against this background, some observations concerning the use of tools should be offered: 7j

1. New technology, including software tools, is not easily transferred. TRW has found that technology transfer frequently requires the transfer of technicians. In the case of NSW, the introduction of new tools may require personal demonstrations, classes, consulting and other personal contact between tool purveyors and tool users, even though the available documentation, both on-line and hardcopy, would seem more than adequate to enable any intelligent, interested person to understand the value of a tool. 7j1

2. Small, easily understood tools have a better chance of early acceptance than large, pervasive tools. Tools which help a user to do that which he has always done are well accepted; tools which require the user to change his philosophy of doing things are less likely to be tried. 7j2

3. The biggest gain available to the user comes from the use of tools which require a change of operating philosophy on the part of the user. 7j3

4. Software tools improve the quality of software. Projects recognizing that they have quality problems will be the best customers. 7j4

5. Tools, like other programs, improve with use. Faults will be removed and better capabilities will be installed. Because NSW will expose tools to more users, sooner than conventional use, the tools will mature sooner. 7j5

These observations concerning the size, acceptability and usefulness of tools suggest that the tools available within the NSW should include both small, easily assimilated tools and large, high payoff systems. 7k

Demonstration of the NSW in a weapon system software environment is possible using only the tools now resident or soon to be resident in the NSW. However, tools characteristic of those now in use in, and specialized to, the weapon system software production environment will be required before a fully representative demonstration can be accomplished. 7l

In an operational environment, the projects relying on the NSW must receive some assurance that the tools which they choose to use will continue to operate in an expected manner. Such assurance can be obtained by the implementation of a configuration management and quality control plan such as that included in Appendix V of this report. Such implementation will require modification of the draft document in order to accommodate the diverse interests and authority centers of the NSW. Discussions leading to the design and implementation of an acceptable plan should be undertaken early in the life of the NSW. 7m

APPENDIX A - ABSTRACTS OF TYPICAL TRW TOOLS 9

The following brief descriptions were selected from the Software Control Facility (SCF), the central repository of tools at TRW. The descriptions were submitted by the authors or custodians of the programs, along with other descriptive material, on preprinted forms which are on file with the programs in the SCF. These tools are not fully representative of the tools within the SCF in that specialized tools such as interpretive computer simulations and engineering analysis tools such as trajectory simulations or circuit design programs are not included. 9a

Unless otherwise stated, all of the following programs are written in FORTRAN for the TRW/TSS system, which is based on the CDC SCOPE system. 9b

HIGH-POWERED ACCOUNTING RESOURCES PROGRAM 9b1

HARP is a computer program designed to handle life cycle costing, and management information and cost control applications, with capability to optionally portray activity descriptions (WBS items, proposal items, project milestones, etc.), activity numbers, schedules, and resource or cost information. HARP is designed for use in assembling and displaying life cycle costs and schedules for various complex projects.

The principal outputs of the HARP system are a detail chart, a cumulation chart, a histogram chart and a summary chart. The detail chart is an integrated visual display of activity description, activity number, schedule and cost information with total time phased cost summations presented on the chart weekly, monthly or yearly as preferred. Cost summations may also be made within category breakouts as desired such as engineering and labor types, computer, travel, reproduction, etc.

Benefits

HARP is in routine operation on the STS Project and has proven to be an effective management tool. Work Breakdown Structures (WBS), cost projection, project schedules, etc., are presented effectively and the system makes it easy to modify the reports as conditions change.

JOVIAL INSTRUMENTATION, GENERAL SYNTAX ANALYZER, AND WORDPROCESSOR 9b2

JIGSAW is a machine independent test tool package written in FORTRAN for structured JOVIAL code. The package consists of three modules: a Pre-processor, a Code Auditor and a Path Analyzer.

The Pre-processor Module accepts and translates structured JOVIAL source code statements into JOVIAL J3. It provides an expansion of many constructs while also having a keyword rejection and macro capability.

The Code Auditor Module provides a simple line-by-line inspection of the code for violations of project standards specified for the JOVIAL and UNIVAC assembly code.

The Path Analysis Module instruments JOVIAL code in order to provide a quantitative assessment of the thoroughness of program testing and determine the frequency of execution for each portion of the program logic during a test series.

Benefits

JIGSAW allows the JOVIAL programmer to code directly in structured constructs. It provides a high level language macro expansion capability and assures that the code will conform to project standards. JIGSAW provides a user oriented Path Analysis capability as well.

SOFTWARE PRICE ESTIMATION AND DISTRIBUTION 9b3

SPREAD (Software Price Estimation and Distribution) is an iterative tool used for developing and analyzing the cost data applicable to any software procurement effort. A historically verified data base is integrated with information peculiar to the project being studied to yield a result geared to the using organization's present or desired methods of doing business, while tempered by the realities of past performance.

SPREAD may be utilized for any of the following tasks:

- Prediction of total costs and labor requirements inherent in a software procurement.
- Spreading of total costs and labor over time phases, among subsets of development activities, and among various accounts within a proposed management structure.
- Analysis of the effects of changes in overhead rate, profit margin, or other critical cost components.
- Scheduling of incremental manpower requirements in a developmental effort composed of several separately scheduled segments.
- Comparison of actual performance values to predicted totals.

SPREAD is coded in the FORTRAN-IV language for use on the TRW Timeshare System (TRW/TSS). The reader is presumed to be familiar with the basic operation of this system.

Benefits

SPREAD is a very useful tool to iterate the cost of software projects. It gets management into the act early with the result that the costs are known before the technical volume is complete and scope cutback can be done before the proposal goes out the door.

SPRINT 9b4

SPRINT is a software package for modeling and evaluating computer hardware and software systems. It is a flexible tool enabling alternate software structures with differing systems concepts, and alternate computer configurations to be rapidly evaluated against the operational requirements.

SPRINT outputs are: (a) load profile graphic listings showing computer throughput limits versus the operational usage of computer resources, (b) internal and external storage, and (c) data transfer bandpass over the selected time period.

The SPRINT program is intended to support short-term proposal efforts in which a software design must be demonstrated to satisfy requirements. If the computer hardware is unspecified, SPRINT will also aid the computer selection function.

Benefits

The input data is that data normally produced during a front end requirements analysis. Final results can be obtained in a short time for a modest cost and can be directly incorporated into a proposal document. The program can be used iteratively in conjunction with an evolving design without any requirements to reprogram.

PROGRAM DESIGN LANGUAGE 9b5

PDL is intended for the production of structured designs in a top-down manner. It is a "pidgin" language in that it uses the vocabulary of one language (i.e. English) and the overall syntax of another (i.e. a structured programming language). Since the language is formalized to a degree it is possible to use a processor to format and cross index the design document as it is being written. The output format is intended to replace flowcharts and to form the body of the design document.

One virtue of PDL is that a rough outline of a design can be quickly constructed. The report format of PDL allows the design to be easily understood by people other than the designer. Thus, criticisms, suggestions, and modifications can be quickly incorporated into the

design, possibly resulting in complete rewrites of major sections. As the design matures more detail can be added in successive passes through the system.

Language

The program is leased from Caine, Farber & Gordon and is available only in absolute form.

Benefits

Dramatic reduction in the time to design and code programs. Reduces computer time required for testing. Reduces documentation costs since it can take the place of flowcharts.

COMMON SPECIFICATIONS STATEMENT GENERATOR 9b6

COMGEN is designed to provide the FORTRAN programmer with automated software for the development and maintenance of computer programs. The prime capabilities of the system are as follows:

a. Generation and automatic insertion of common specification statements into FORTRAN programs.

b. Determination and display of all program common variables and common variables per subroutine.

C. Automated operations for converting existing programs over to a COMGEN compatible data base for future updates and maintenance.

d. Output of variable and subroutine cross-references.

e. Definition and display of subroutine interfaces.

f. Provision of numerous automatic programmer aids.

Benefits

Protection and control of the data base as well as ease of entering data base changes.

FLOWGEN 9b7

FLOWGEN generates program flowcharts automatically from FORTRAN source cards. Comments which occur on the source code are transferred to the flowchart in an intelligent manner. The flowcharts produced are divided into pages which can be photoreduced and printed on 8 1/2 X 11 inch paper connectors, continuation remarks, and page numbers are automatically inserted:

Benefits

It represents a cost savings if detailed program documentation is required. It actually represents the as-built code and is up to date.

COMPREHENSIVE AUTOMATED DOCUMENT MAINTENANCE AND UPDATE SYSTEM 9b8

CADMUS processes source decks to extract program documentation. Comment cards, identified by keywords, are extracted from the source file and reformatted for printing. A Table of Contents is also prepared. In addition, cross indices of program libraries, and charts of module cross references are prepared.

Benefits

Estimated savings of a person half time. Additional output is provided that would never have been attempted, otherwise.

INTEGRATED JOVIAL SUPPORT TOOLS 9b9

IJST provides the capability to improve the readability of a source program written in JOVIAL J3 by reformatting the statements into a standard form.

IJST provides the capability to develop a structured JOVIAL J3 source program through the user of "structured" macro statements (e.g. WHILE, IF-FOR). IJST translates these macros into structured JOVIAL J3 code blocks.

IJST provides the capability to audit a JOVIAL J3 source program by analyzing the program for violation of coding standards and conventions.

IJST provides the capability to analyze the performance of program written in JOVIAL J3. IJST indicates the amount of code being executed, how often a segment of code executed, and the threads, as the program executed.

IJST executes in the time-share mode on the TRW/TSS as well as the batch mode. It consists of three separate programs.

SET AND USE OF ROUTINE VARIABLES ANALYSIS PROGRAM 9b10

The SURVAYOR Program in an automated software verification tool which will perform the necessary structural path analysis to identify:

- All possible paths in a routine.
- Where in a routine variables are computed and used.

Based on path and usage data collected, the SURVAYOR Program will:

- o Evaluate each local variable to identify:

  which are computed and not used

  which are preset and not used

  which are used in computations without being given a value

  the path for which a variable may be used without being given a value.

- o Evaluate each global variable to identify:
  equivalences not required
  common blocks not required
  subroutine parameter variables not required
  unreferenced external definitions.

- o Provide detailed information on the quantity and type of software processed.

Benefits

SURVAYOR will identify static path and data errors at the routine level which may not be caught by the compiler.

STRUCT 9b11

STRUCT audits FORTRAN source code for adherence to structured programming standards. Allowable segment structures include: sequence, IF - THEN - ELSE, DO UNTIL, DO WHILE, CASE, and ESCAPE. Outputs include the PACE segment transfer table for each subroutine, and a management summary of all routines for the whole program. Input for this program is not source code, but a segment transfer table generated by PACE.

Benefits

Language independent, i.e., runs off segment transfer table, not source code.

DOCEDT 9b12

DOCEDT provides the capability to generate and maintain a document on tape. The document is initially generated on punched cards, and then it is maintained on tape with corrections and additions incorporated by the use of DOCEDT change cards.

Benefits

This tool can significantly reduce the cost of generating and updating software documentation.

COMPASS CODE AUDITOR 9b13

The COMPASS CODE AUDITOR audits CDC COMPASS code for adherence to a predefined set of 12 software standards. Some of the standards deal in explicit key phrases in the preface commentary block. Outputs include a list of standards against which the program audits, followed by each routine with deviations annotated in-line, wrap-up summaries of deviations following each routine, and a good summary at the end.

Benefits

Audits large programs in seconds; prints detail and/or management summary reports.

FORTRAN CODE AUDITOR 9b14

The FORTRAN CODE AUDITOR Program examines FORTRAN source code for adherence to a predefined set of 31 software standards. It was developed for use on the Systems Technology Program, but most standards can be applied to any FORTRAN program. Outputs include a list of standards against which the program audits, a listing of the user's source code with deviations flagged, a summary of deviations after each subroutine, and a summary for the entire program.

Benefits

Audits large libraries of FORTRAN source code in just seconds. Provides detail and/or management summary reports.

PRODUCT ASSURANCE CONFIDENCE EVALUATION 9b15

PACE has two main functions: it performs a static analysis of the user's FORTRAN source program, and it instruments the user's code so that the user can monitor test execution to measure test sufficiency. Outputs from the static analysis include: source code annotated with segment numbers, a program structure summary, a segment transfer table, and a segment description table. The output from the instrumentation module includes, in addition to the user's normal test output, a test execution frequency table showing the number of times each segment was tested and a list of all unexecuted segments.

Benefits

Only available tool on Systems Technology Program to measure test sufficiency; only available tool to act as driver for STRUCT.

NODE DETERMINATION AND ANALYSIS PROGRAM 9b16

NODAL is designed to aid the user in executing all the source code and all the branches in a Fortran program. It analyzes the code of an existing program and instruments or adds code that will record the execution of the program's nodes. A node is defined as an entry point to the smallest set of consecutively executable statements to which control can be given during program execution.

At the normal end of an execution of the user's instrumented program, NODAL will obtain control and provide information about the frequency of execution of each node. Also provided is a test effectiveness ratio (nodes executed/nodes identified) for each routine, a test effectiveness ratio for the entire program, and a list of the program nodes not executed.

AUTOMATED TEST, COMPARE AND MONITOR PROGRAM 9b17

Designed for use in a controlled I/O environment such as access to a global data base. It may be used for development testing and final checkout of unit source code. A unit of code may consist of a control routine and a number of subroutines comprising a functional module. Any number of consecutive test cases may be executed in a single run. Test cases may be isolated or information may be transmitted to successive cases if desired. The results of each test case is compared to expected values and discrepancies are flagged. Initial, final and expected values are displayed for each discrepancy. The statement label execution path is displayed for each test case. At runs end, a summary is displayed indicating the number of executions of each statement label flagging out unexpected labels. Also, a summary of the test cases indicating which, if any, contained errors.

This is similar to the AUTORESET tool available on the IBM 360 computers.

Benefits

Controlled test environment for unit code. Uniform test integrity prior to system integration. Eliminates retest blues. Automated I/O and test documentation. One run does it all. Permits thorough modular checkout. Cost and schedule savings due to early detection and correction of software errors.

AUTORETEST 9b18

The principal application of the AUTORETEST program is the automation of user software test results re-validation. The system provides an automated comparison between selected old and new test parameters thereby allowing invaluable documentation of the test cases. This

system also provides a flexibility in that a tolerance criterion may be assigned to each comparison and, thereby, suppress insignificant differences.

This is similar to the driver tool available for the CDC computers.

Benefits

AUTORETEST is easy to implement, only uses 4K of core and has a very small impact on user program execution time. It is invaluable where test results are too voluminous to be manually verified.

CHECKSUM 9b19

CHECKSUM can be applied to a program library following acceptance testing to establish a baseline of individual program checksums. CHECKSUM will identify additions to the library, deletions from the library and changes to old members of the library. It identifies intentional and surreptitious modifications to a controlled system and provides a concise means of describing a software configuration.

Benefits

CHECKSUM provides an automated accurate and efficient way of maintaining cognizance of a software configuration at all levels (subroutine, module, program, and in all areas (data, code, documentation, etc.)).

GENERALIZED INFORMATION MANAGEMENT SYSTEM 9b20

GIM is an on-line generalized data management system. The principal GIM features which make it a valuable tool for applications-like configuration management are:

- o No programming is required for application.
- o Files can be constructed, loaded with data, updated, or accessed from a terminal or in batch mode.
- o It is user-oriented. It can be queried with an English-like user language, it can retrieve selected data and it can generate reports at the terminal.

GIM is being used for configuration management on the Systems Technology Program (STP) and for all software configuration management at the Sunnyvale Lab of TRW.

Benefits

A generalized information management system with data storage retrieval and reporting capabilities. This on-line or batch mode system has been used extensively by TRW to maintain and report project and configuration management records.

Language

PWS and IBM Assembler

Machine

IBM 360-370

APPENDIX B - JAVS APPRAISAL 10

INTRODUCTION 10a

The following information is an analysis of the JOVIAL Automated Verification System (JAVS), developed by the General Research Corporation for Rome Air Development Center, contract number F30602-73-C-0344. This analysis is based upon: 10a1

A detailed study of the following documents:

1. Methodology for Comprehensive Software Testing: General Research Corporation, June 1975.

2. JAVS Computer Program Documentation: User's Guide, General Research Corporation, November 1975.

3. JAVS Computer Program Documentation: Reference Manual, General Research Corporation, November 1975.

Information obtained in a three day JAVS training course presented by General Research Corporation at the Rome Air Development Center during June 1976.

JAVS CAPABILITIES 10b

JAVS is a system of automated tools intended to be applied during program testing, to aid in the verification of untested program paths and the identification of test cases that will improved testing coverage. This is provided by analysis of program structures, instrumentation of the system through insertion of appropriate software probes to measure testing coverage, and extensive reports which identify paths in the program structure that remain to be exercised. 10b1

JAVS is organized into a series of independent processing steps, each performing a unique task that communicates through a common data base called the library. The various steps and their basic functions are: 10b2

A. Source Text Analysis

Source text input lexical analysis;

Initial source library creation.

B. Structural Analysis

Execution path identification;

Library update with structural and path information.

C. Module Instrumentation

Program instrumentation for path coverage analysis and program performance directed by user;

Library update with probes instrumentation.

D. Test Execution

Execution of instrumented code;

Analysis of directed program performance.

E. Test Effectiveness Measurement

Detailed analysis of program path coverage;

Execution traces and summary statistics.

F. Module Testing Assistance and Segment Analysis

Testing assistance for improved program coverage.

G. Retesting Guidance and Analysis

Retesting requirements analysis for changed modules.

Included within these standard processing steps is a data management system, which provides for library merging, module selection, initialization, and print-outs of library contents. 10b3

The JAVS processing steps are controlled by a command language which allows selection of various processing options within each step. The user is also allowed to repeatedly execute certain sequences of commands for program modules contained within the library file. Certain commands may be used in all steps, while other commands are process dependent and are recognized only when employed within that step. 10b4

A. Source Text Analysis

This step performs the initial analysis of JOVIAL source language statements, transforming them into a format appropriate for storage on the library file.

The primary output from this step will be a new library file containing the transformed source text with each START-TERM sequence separated into distinct modules. Printed output will consist of a listing of the source text and a summary of the contents of the library file.

B. Structural Analysis

This step performs an analysis of the structure of the modules on the library file produced during source text analysis. The structured data is then added to the existing library data and an updated library file is created. Program structure is defined in terms of decision to decision paths (DD-Paths). A DD-Path is a set of statements in a module which is executed as the result of the evaluation of a conditional statement within the module up to but not including the next conditional statement.

The primary output of this structural analysis processing step is an altered library file. Printed output consists of the structural analysis results. Any errors encountered during structural analysis are also printed.

C. Module Instrumentation

This step performs the instrumentation of modules which have been structurally analyzed. The primary function of this step is the automatic insertion of a set of probe statements into each module. The instrumented modules are logically identical to the original source code. There are two types of instrumentation available:

A. Structural Instrumentation

Software probes are inserted into the source code at the start of each procedure, at each return point, and at each statement which begins a DD-Path.

Each probe includes a call to a special auditing routine which records information concerning flow of control in the executing modules.

B. Directive Instrumentation

Software probes which monitor the results of assignment and exchange statements during test execution are inserted in the source text where the user has placed JAVS instrumentation directives. Each directive controls execution time output and is interspersed with normal program output.

The primary result of instrumentation is a modified library file. Printed output is used to inform the user of the extent of the instrumentation performed. The actual instrumented source code can be printed and punched in this step (or in subsequent steps) by using appropriate commands.

D. Test Execution

In order to proceed with verification of the software system, the instrumented source code is compiled with a COMPOOL, which supplies definitions for the JAVS data collection procedures.

The compiled code is loaded with any required external modules and the data collection procedures from the JAVS object code library and is executed. During test execution, the program operates normally, reading its own data and writing its own output. The instrumented module calls the data collection routine, which records execution trace information on a trace file and accumulates data on module and DD-Path usage. Performance data resulting from JAVS instrumentation directives are interspersed with normal program printed output. Each test execution may consist of a number of test cases. The user identifies the start and end of all test cases by executing a call to one of the data collection routines. These identification calls are manually inserted into the program prior to compilation. All other instrumentation of the source code is automatically performed by JAVS.

E. Test Effectiveness Measurement

This step analyzes testing coverage by utilizing the module structure information available from the library file. DD-Paths are related to the test coverage, and user selection of execution paths for further testing can be determined.

One of the reports produced shows the cumulative frequency of execution for each statement during the processing of test cases. This report is produced for each specified module. A report is also available which shows a summary of the time in milliseconds spent in execution for a specified module during all of the test case executions.

F. Module Testing Assistance and Segment Analysis

When sets of untested DD-Paths have been identified, the user can obtain assistance in the generation of appropriate test cases to execute these DD-Paths.

G. Retesting Guidance and Analysis

This step produces a set of analysis procedures that operate on the interdependencies between modules in the library file. Some of the capabilities of this analysis are oriented toward providing an insight into overall program structure. The intermodule dependencies together with test execution reports may serve to identify the retesting requirements for changed modules.

H. Performance Testing with JAVS Instrumentation Directives

The capabilities of the JAVS instrumentation directives provide the means for monitoring the performance of the program without affecting the logic.

Instrumentation directives are inserted by the user in the JOVIAL source code before it is processed by JAVS. Since each directive is a special form of JOVIAL comment, it is ignored by the JOVIAL compiler. The source text analysis recognizes the directive; the module instrumentation step then analyzes the directive and provides additional instrumentation as a user option. When the probed text is executed, printed output from these directives is interspersed with normal program output. There are four (4) JAVS instrumentation directives: ASSERT, which monitors the true value of a particular logical condition; EXPECT, which monitors the value range for selected variables; and TRACE and OFF-TRACE, which monitor the effect of computations upon one or more variables.

JAVS EVALUATION 10c

Evaluation Objective 10c1

The objective of a state-of-the-art verification system should be to operate with sufficient economy as to allow system level testing on medium to large scale software. This evaluation attempts to assess the JAVS V&V system within the framework set by this objective. We set several evaluation criteria intended to help assess such a V&V capability, and then proceed to critique the JAVS system.

Evaluation Criteria 10c2

The criteria used to evaluate JAVS are:

A. Degree of correlation of verification results and the user's high order source language,

B. Amount of overhead to execute the user's program in a verification environment,

C. Cost of the analysis of the user's source language and the preparation of reports,

D. The ability of the system to support V&V over several test executions, and

E. The cost of interfacing with the system (e.g. run setup cost, ease of use of the command language, etc.).

These criteria are clearly divided into two categories: those relating to system capabilities, and those relating to the user interface with

the system. For each category both the costs and benefits of JAVS must be evaluated.

Due to the evolving nature and the limited use made of the JAVS system, much of the evaluation of criteria related to JAVS performance must be deferred until actual operational data is collected.

Evaluation 10c3

This evaluation considers the JAVS capabilities and its user interface. Capabilities are considered to consist of the functions supported by JAVS, the graph theory considerations underlying JAVS, and the techniques used to implement the JAVS instrumentation of the JOVIAL source program. The user interface is considered to consist of the JAVS command language, the means to utilize JAVS in the context of a large system V&V effort, and the clarity and utility of the output of JAVS.

Capabilities

As discussed in the preceding section on JAVS capabilities, JAVS provides the capabilities expected from an automated verification system consistent with state-of-the-art technology. It includes the basic capability of test performance measurement plus the analytic capability to determine test effectiveness, aid retest, and aid testing of interrelated modules. Within this set of capabilities, JAVS includes some leading edge concepts, particularly the ASSERT statement capability.

All automated verification systems that operate on the structure of the source program, as does JAVS, rely on the construction of a graph model of the program. JAVS does not construct the conventional graph representation commonly found in such systems. Instead, JAVS relies on the concept of a DD-Path, which is defined as:

> The smallest executable piece of a program, the DD-Path, is the sequence of activities the program performs in using the outcome of a decision to determine the program's future course of action.

We found the use of this concept to analyze the structure of programs to be a significant drawback of JAVS. Using DD-Paths, there is no direct correlation between a DD-Path and a line of executable code. A single executable statement may be contained within many DD-Paths, and a single DD-Path may contain many extremely disjoint executable statements.

The DD-Path segmentation scheme is an unnatural logical division of the program for most users. A single DD-Path may be much longer than the author's original concept of flow when writing the program. The path may, as a result, contain many shared abstractions which are not

easily correlated with the original program design. In addition to being at odds with the current structured programming concept of nested levels of abstraction, DD-Paths are not easily determined by manual methods.

JAVS instruments source code by the insertion of a call to an auditing procedure at the start of every DD-Path. The procedure call has an argument list containing three arguments which are written to a mass storage trace file along with other accounting data. It is clear that the use of JAVS for the V&V of medium to large scale systems will generate a trace file containing a large number of records if the entire system is instrumented. We believe that the overhead implied by the creation of a file of that size effectively dictates that JAVS must be used selectively for V&V at the system test level.

Data accumulated by TRW indicates that the core overhead induced by the instrumentation of source code in the manner utilized by JAVS may be quite high, often in excess of 100%. Although this overhead is acceptable in a virtual memory environment, it is often unacceptable in the constrained environments surrounding current weapon system software.

We discovered another problem in the JAVS instrumentation methodology relating to the accumulation of module execution times. JAVS contains a post-processing option to print a summary of the time spent in execution within user-specified modules. There are two problems in the implementation of this feature within JAVS:

1. Timing data is collected during test execution for all modules regardless of whether or not the corresponding summary print has been requested. Clock interrogation is typically an operating system function, and may require more than one millisecond of processor time per access. For systems with total module execution frequencies in the hundreds of thousands, as occurs in medium to large scale systems, the time overhead required to implement this feature is prohibitive. The feature is desirable, nevertheless, and means should be found to reduce the associated overhead.

2. JAVS collects timing data at the entrance to and exit from each module. As a result, there is no way to determine the actual time spent executing statements within that module since the module may itself call system functions or other modules. JAVS is unable to back out these nested timings in order to determine correct module timings.

User Interface

The JAVS command language contains more than 70 different

instructions. This language provides the user with the ability to direct and control each processing step of the verification. The commands and their descriptions are presented in two volumes, the JAVS User's Guide and the JAVS Reference Manual. The User's Guide provides a minimal instruction set and demonstrates the basic principles by which JAVS provides services to the user. The JAVS Reference Manual describes in detail each of the JAVS commands, JAVS processing, and JAVS output. Given the volume of commands to be described, the two manuals are well organized and present the information in a logical, user-oriented fashion.

The JAVS output, however, lacks the well thought out clarity of the JAVS documentation. This is particularly true for users unfamiliar with automated verification systems. Because of its orientation around the DD-Path concept, JAVS requires the printing of a number of different reports in order to define the statements associated with the different DD-Paths. It was demonstrated at the JAVS training classes that it is difficult for most users, including users with prior JAVS experience, to determine the DD-Path organization of a given module.

The determination of the dynamic execution frequency of a statement is also difficult with the outputs provided by JAVS. Statement execution coverage information resolved down to the individual executable statement is difficult to obtain as well; significant manual analysis is currently required to extract this information. These problems are symptomatic of the fact that the majority of the JAVS reports present verification results at the DD-Path level and require manual correlation to any other working level.

The evaluation of JAVS in the context of medium to large scale system V&V must also consider the aspects of usage of the JAVS system. The verification of system level software requires that the system under test be executed a number of different times. This requirement arises from a number of different conditions, including the test of systems not designed to process batches of input data serially, excess computer resources being required to run the complete series of test cases at one time, or the correct execution of a given group of test cases causing system termination. Under any of these conditions it is necessary that the verification system be capable of the accumulation of verification results over a sequence of system executions.

Recommendations 10c4

We have found JAVS to be an advanced automated verification system. JAVS does, however, incorporate significant overhead in terms of both human and computer resources which we judge unacceptable in a production tool. If JAVS is considered to be a prototype tool which will be

refined into a production version, we suggest that the modifications described in this section be incorporated. If JAVS is to be primarily a research tool, its present facilities are well chosen, and the modifications should be considered as enhancements.

1. The DD-Path concept should be replaced by conventional basic block techniques which are closely related to the actual graph structure of the module. In this definition, a basic block is (informally) the smallest set of consecutively executable statements to which control can be given during program execution. The first statement of every basic block is addressable, and the last statement of the basic block is the one and only exit from the block.

None of the statements of a basic block may be executed unless every statement of that block is executed. The number of segments and DD-Paths are equal, but the basic block has the advantage that each and every statement is contained within one and only one basic block. The basic block concept thus more closely follows the design of the software and good structured programming practice, and therefore better aids the user in relating execution results to the actual software. Furthermore, the number of reports required to convey the necessary verification information is reduced.

2. The overhead caused by the technique of recording execution monitoring data on a file must be reduced. One technique which may be used is to record frequency of execution data in core. The need for a trace file is thus obviated unless trace data is required; that capability is only required infrequently after testing ascends to the system level. This approach has the additional advantage that the accumulation of data from run to run becomes a trivial problem.

APPENDIX C - NLS APPRAISAL 11

SUMMARY 11a

This report covers the NLS training course given 15-16 July and the NLS Application Discussions of 19-20 July, 1976. The report concludes with observations based on further experimentation with NLS. 11a1

NLS TRAINING COURSE OBSERVATIONS 11b

Course-Specific Observations 11b1

The course began with a statement that the goal of the course is to provide training in the NLS basics: typing text online, editing that text, and handling messages with TENEX and NLS. This statement is important in that it defined the scope of the material to be covered. As the course progressed, however, it became apparent that for many users introduction of the message-related material is premature in the first course. The course was stated to contain material essential for both DNLS and TNLS. DNLS is Display NLS (CRT and cursor) and TNLS is Typewriter NLS (any teletype compatible terminal).

The course proceeded immediately into presentation, reading, and discussion of the TNLS introductory course, which itself begins with connecting to the computer and creating files.

This approach to the course can be improved, in that it fails to provide the student with a framework of thought with which to deal with NLS. The student is therefore required to provide his own framework; for programmers and other users completely familiar with computer driven text editors this presents no problem. For programmers used to batch systems, or for clerical personnel with limited or no experience with such systems, a discussion of the following sort is important. (Note: The following discussion is a paraphrase of one given between the two days of training to the TRW secretary in attendance at the course. It proved to be of significant value in increasing her understanding of the course material.)

> "A document is fundamentally a string of characters. This statement is particularly true if typewriter keys such as carriage return and tab are considered to be characters. At the simplest level, a computer text editor could store a document in that manner, as a single, undifferentiated string of characters.
>
> The problem with such a representation is that it is extremely difficult to deal with for editing purposes. One way of solving this problem is to divide the document into lines of text (not necessarily related to the lines of the finished document), and to provide some readily understandable 'name' for each line. In this way the size of

the conceptual pieces of the document is reduced, thus facilitating the editing process. Line names are typically formed by sequentially numbering the lines of the document.

Many text editors use exactly that representation scheme. The TRW MPS system and the Proprietary Computer Services PCS/TEXT (an adaptation of IBM's ATS) are examples of this type of text editor, as are the many different implementations of BASIC.

A document is more organized than a simple sequence of lines. Typically a document is written following an outline, and thus has a kind of a two-dimensional structure. The design of NLS recognizes this fact, and allows the user to specify structure within the document. In an outline, the smallest unit is typically the paragraph; the corresponding term in NLS is the statement. NLS statements may include from one character to an entire paragraph. The structural representation of NLS permits the user to deal with the structure contained in the outline of the document.

Finally, the entire collection of information the user deals with, from the long string of characters in our simple representation to the complex structural entity handled by NLS, is stored by the computer in something called a file. The file may be thought of as the container in which the document is stored. Every file has a name, which is the name by which the computer knows the document. Retrieval of the document from the computer storage is done by means of this file name."

The training course could benefit from either some printed or online self-study material which would be worked over by each student prior to beginning the course. An example of the type of material intended is the NLS Secretarial Functions Guide (26442). By providing the student with some introductory experience gained at his own speed, the level of understanding (and therefore the rate of comprehension) of the student would be increased. Furthermore, the initial nervousness and hesitancy encountered by many persons when faced with the online computer would be partially overcome.

Another important consideration with respect to NLS training is that a serious attempt must be made to judge the level of sophistication of each student and to group students into classes of approximately equal ability. This requirement is more pronounced in the case of NLS than with other technical training courses because the level of involvement and interaction required from each student is higher due to the introduction of the online system into the training process. Violation of this principle will result in boredom from the advanced students and confusion in the slower ones. Neither effect is beneficial.

The format and structure of the training classes is not conducive

towards question/answer interaction between trainer and student. This effect is certainly not in any way the fault of the SRI trainer, who did an excellent job of adapting to the difficult situation of having three people to train, all of different levels of sophistication and having differing goals with respect to the course. Instead, we believe the effect is caused by these factors:

1. The "pre-planned" nature of the course, wherein the material is taught from an outline, tends to inhibit deviation from the outline to discuss topics in richer depth.

2. The use of the NLS system online during the course is based on a series of pre-planned exercises; using the system as an aid in answering questions requires the impromptu generation of an example by the trainer. For difficult questions, such examples are often either not easily forthcoming or require (for demonstration) knowledge exceeding the students grasp.

The problem of barriers to questions/answers is less, of course, with aggressive students. We would also expect the problem to diminish as the student's level of sophistication increases.

The size of an NLS training class is necessarily limited due to the close interaction required between student and trainer and due to the use of NLS online. This implies that training to kick off a large programming and/or documentation project could become a serious problem. One approach to solving the problem (i.e. an approach to increasing the feasible, effective class size) might be to automate some of the instruction in the form of an NLS subsystem. This teaching subsystem would lead the student through the material. Interaction with an instructor would be required primarily to answer questions and to initiate the course. Such a subsystem might well be patterned after the successful computer aided instruction work done on the PLATO system at the University of Illinois.

Underscoring this commentary about class size, questions, and answers, it is important to recognize that the questions that were asked during the two-day training sessions contributed materially to our understanding of and facility with NLS.

Another consideration which may increase the size of the class a trainer can handle is that some time should be allowed in the training for solitary practice with the concepts and commands learned. Such practice time should occur sufficiently often that the number of concepts new for that practice session is not excessive. Modification of the training course in this manner is in some ways a partial step down the road towards the teaching subsystem suggested earlier.

Other Observations 11b2

SRI strongly suggests that every organization using NLS identify someone as their NLS "architect". In their view, the architect at least is responsible for answering questions, in-house training of users, and interface between the installation and SRI.

Effective use of NLS requires more than a TI terminal and a 300 bps modem. Use of the Display NLS Workstation requires an ARPANET link, or other telephone type linkage, to an NLS machine capable of data rates greater than or equal to 1200 bps. Furthermore, SRI stated that use of the Display Workstation does not remove the need for hardcopy. A local hardcopy printer is therefore necessary. (The alternative, of course, is to have a local host running NLS; the printer is then a peripheral of the host, and terminals may be hardwired to the host and therefore operate at higher rates.)

The number of Workstations available for use with NLS will be an important factor in the results of any experiments (or work, for that matter) using NSW or NLS. The average number of users per workstation is apparently about four at SRI and many other NLS installations. (At TRW, the OTH project uses eight terminals of similar capability to service approximately thirty two programmers, administrators and secretaries.) It also seemed clear that the Workstation, with its split-screen capability, would be of value to a programmer for insertion of previously coded procedures and for debugging.

In the online programming environment (UNIVAC EDITOR) in the OTH project at TRW, the ratio of four users to one terminal causes terminal contention by the secretaries, programmers, testers and other interested parties. It is not clear at this time whether the richer capabilities of NLS will enable the users to do their job more efficiently and thus require less terminal time, or that the richer environment will cause each user to put more of his work online, and thus require more terminal time. We tend to believe that the latter course will ensue.

If we are right, the ratio of users to terminals will be found to be properly lower than four to one. It is not clear that all of the terminals should be Display Workstations, however, since many tasks are done as well or better with a hardcopy terminal, at a saving in terminal and computer cost.

The impact of this argument is that a sizable capital investment will be required for projects of anything more than minimal size. If a local NLS host is also required (more on that later), then the investment is an even larger problem.

Two other miscellaneous observations:

First, the level-oriented structure provided by NLS is ideal for indented, structured programming, and represents a solid and novel

solution to the problem of changing structure within a program as the control structure is changed (which usually fouls up the indenting). SRI uses NLS in this way for their own programming, and seems to have developed some conventions which contribute substantially to the program development process in the context of structured programming.

Secondly, although much of the command and other information on the NLS quick reference card is soon memorized, the card itself is somewhat inconvenient. We recommend the printing of the quick reference cards as wall charts to be hung over the workstation area.

APPLICATIONS DISCUSSION NOTES AND OBSERVATIONS 11c

SRI noted that there are three major problems involved in bringing a user organization onto NLS: 11c1

1. NLS is a complex technology product.

2. The "style" of each individual using the system is different. Such personal effects can often block adoption of the system.

3. Management commitment is required to see the process over the bumps and hurdles until NLS is accepted and can begin to produce operational benefits. (Note: for commitment, read "will" and "resources".)

SRI further stated their motives to be as follows: 11c2

1. To deliver the technology to the outside world.

2. To obtain feedback about the system which will help guide its improvement.

3. To build a community of users, organizations, and NLS architects which will be able to share concepts, ideas, and products.

SRI provided these observations about costs. Bell Telephone averaged $10.00 per connect hour; RADC averages $30.00. A general average is believed to be $25-30. SRI's philosophy regarding the sale of machine time through OFFICE-1 is that the service is intended to get users introduced to NLS. When a significant load evolves the user is encouraged to acquire his own facility with NLS installed there. NLS costs on a users facility (i.e. SRI support) are approximately $70K per year. A minimal size host to run NLS was estimated to cost $400-500K. SRI believes the typical gestation before an installation is ready to acquire its own host is about 2 years. 11c3

TRW USAGE OF NLS 11d

In order to gain further insight into the benefits and problems of using

NLS in a weapon system atmosphere, several projects at TRW have been accomplished or aided by NLS. 11d1

- o A document for the Minuteman project concerning next generation ICBM guidance requirements was initiated. The effort was prematurely terminated when the project was physically transferred to Norton AFB from Space Park.

- o A document concerning the requirements for a Higher Order Language with which to construct the next generation of ICBM guidance software was completed by an NLS trained secretary and an "untrained" software engineer. The engineer created approximately one third of the document online, and edited all of it. His training was entirely informal.

- o A FORTRAN program was written in NLS and submitted directly to the DEC compiler. The code was modified, debugged and used without any problems.

- o The entire C-5 design document for the Correlation / Identification component of the OTH-B project was transferred from UNIVAC Exec-8 to NLS, and JOVIAL code was produced in NLS. The code was not compiled because no suitable structured preprocessor and no compiler were available.

- o A variety of output devices was successfully used to obtain printed copy. Of particular interest was the Xerox Graphic Printer at ISI which was used to provide COM formated output for an IR&D project report.

- o This report was generated using NLS.

In the six month period that TRW has used its own display work station, problems related to the equipment or the ARPANET occurred. 11d2

The most significant problem concerned the loss of the ISI TIP, and concomitant hardware and software failures on the ISI "C" machine which became our ARPANET access port. For most of a month we were restricted to 300 bps ports on the USC TIP, which severely restricted our ability to use DNLS. At that low rate, the TNLS system seemed to be more productive.

A perplexing problem in which the character j was substituted for the character p on a random basis (frequency higher than 50%, sometimes) was traced to a bad telephone line. Simple redialing at hookup time has always solved the problem, which continues to occur.

The TRW modem at ISI failed. Since its failure was in its ability to "answer the phone" we, after consultation with the manufacturer, exchanged our two modems and now have the faulty modem located at the terminal, where it causes no problem.

RECOMMENDATIONS

Problems In Using NLS

Because the NLS is a very large, complex software system, it must be experienced in order to be appreciated. Once appreciated, it can be used productively by those who also understand the environment of its use. For that reason, a new user can expect to be required to interpret the NLS capability into his own terms, and will not, in general, find a manual which explains its step by step use in his environment. Some examples of the kinds of problems which TRW encountered in its use of the system may be found in the following paragraphs.

A problem for TRW concerned the use of the formatting commands and the output processor. We found that to modify the two FORMAT general purpose styles required a surprising amount of time, part of which was due to the fact that the preset values of some of the parameters were different than the documentation indicated. Eventually, we were able to generate a simple format of our own devise, which uses very few format commands, from which we can deviate with a high degree of predictability. A fair diagnosis of our problem might be that we never should have used the FORMAT subsystem in the first place.

Although the FORMAT generated styles involve verbose command insertions, they do not provide for footnote insertion, which must be done at the command level. Footnotes must include the page number generation, and must be reset by the user to page number generation "on the next page", which location is not known until print time, and can vary from printing to printing if text is added or if spacing is changed.

A more fundamental problem which TRW noted in using NLS concerned super and subscripts, for which there is no provision. This affords a wide area of compatibility in choice of terminal, but can be expected to cause problems for mathematically based high technology areas. TRW has encountered this problem while using other word processing systems and does not consider the problem to be more than very annoying. The traditional workaround is to leave space for equations, which are inserted by hand before reproduction. A slight variation is to present all equations in a single list which may be appended to the document. Neither of these solutions is of much value to the online worker, however. A further workaround might be to express equations in FORTRAN.

A related problem is the lack of underline capability, which is specifically required in many military style manuals.

These "problems" which TRW encountered in its usage of NLS were judged to be minor, since each was found to have its solution, once the nature of the problem was perceived. They probably serve to illustrate the kinds of pitfalls which await the new user of this kind of technology.

Some inefficiencies or faults in the system were noted by TRW. These were primarily related to the little used LPPRINT subsystem and to the excessive number of meaningless messages which appear briefly on the screen during some processes. The problem with LPPRINT concerned its insistence on inserting page eject commands into the print stream such that the printed text could not be properly paginated. This problem has been corrected by the distribution of a totally new Copy Printer subsystem called LPOUTPUT. The meaningless message problem appears to be only an inefficiency, and has been corrected in the experimental version of NLS (NLS9).

NLS Training 11e2

Much of the difficulty with NLS training that SRI seems to encounter (and TRW would expect to encounter in an extensive training exercise) might be greatly reduced by a change in philosophy of presentation. Presently, all documentation and classroom instruction is in the order: TNLS and then DNLS. TRW suspects that the order should be reversed, that the highly visual, rapid reward, situation available to the DNLS Display Workstation should be used to teach the fundamentals of NLS and the relatively obscure TNLS reserved for intermediate students.

Such a course would best be accompanied by an ordered set of DNLS course outline manuals which make no reference to TNLS. A final step would then be required to familiarize the student with the techniques necessary to do without the display terminal.

NSW Experiment 11e3

Experimental Environment

The TRW Unit Development Folder (UDF) software development technology currently being successfully applied to the System Technology Program and OTH-B projects has conclusively demonstrated that the essence of a software development effort is the collection of information extant at any given point of time for each component of the system. In a predominately batch environment, this collection of information, the UDF, is maintained as a notebook by the responsible programmer or engineer. In an online environment, however (as is the NSW), the UDF clearly should be maintained in online storage. Any hardcopy of the UDF that might be made must be recognized as a snapshot at a particular point in time, and not as a surrogate for the real UDF maintained in computer storage.

This argument leads to the conclusion that the NSW tool used to directly maintain (as opposed to tools used to operate on) the UDF is, in fact, the central tool of the entire system, and is the one with the maximum leverage to make the NSW a success or a failure.

This central tool is, of course, a text editor (for lack of a better term to describe the extended capabilities provided by NLS). One alternative for the TRW NSW experiment is, therefore, to center the experiment about a text editor, and to determine how well that tool supports the weapon system software development environment. A necessary prelude to the experiment would be the selection of the subject text editor from those to be available on NSW. Based on the NLS capabilities and a basic understanding of the capabilities of other text editors, we would recommend NLS as the subject of the experiment. One particularly strong deciding factor is the open-endedness of NLS, which would allow the system to be tailored to support an online adaptation of the UDF technology. This idea is discussed further in a later paragraph.

Nature of Experiment

One very interesting idea for an NSW experiment, incorporating the assumption that an NLS-oriented experiment is a good equivalent, is based on the availability of the MIL STD 490-C-5 specification for the OTH-B Correlation and Identification (C/I) module on the Univac 1110 machine. The experiment would (at a high level) consist of the following steps:

1. Transfer the C-5 spec from the Univac system onto NLS. Aside from being a necessary prelude to the remainder of the experiment, this step will provide information on the difficulty of adapting externally machine readable text and programs into the NLS system. That information will be valuable when the time comes to determine the costs of transferring an on-going project onto NSW.

   This step was accomplished in cooperation with SRI-ARC and ISI. Problems were encountered by ISI in reading a seven track tape written by the UNIVAC 1110 EXEC-8 system, and additional problems were encountered by ARC because the 1110 print file contained no carriage returns, just twenty word logical records for each line. Both problems were judged to be solvable, with some research and experimentation required. Up to four person weeks would be allowed for such a future effort.

2. Code the C/I module in JOVIAL from the online C-5 spec, maintaining the design and code for each Jovial PROCEDURE together with the associated test cases and (perhaps) test results.

3. Debug the complete C/I module. This will provide important data on the costs, advantages, disadvantages, and schedules required for NSW-based software development. It will further address the issue of test drivers in the context of the NSW.

4. Offload the C/I module from the NSW back onto the Univac 1110,

and integrate it into the rest of the OTH-B software system. This step will serve to validate that software developed with NSW is "real world" software, usable in a live weapon system software environment.

APPENDIX D - EDITORS ON NSW 12

INTRODUCTION 12a

This is a study of the requirements for text editing capabilities in the context of the NSW, with particular emphasis on the kinds of text editors available on the NSW. It adopts the point of view of the weapon system software producer. 12a1

The first question that such a study should answer is why another text editor survey or study should be conducted in light of the many and varied ones already published, notably that of Van Dam and Rice. The answer to the question is that the power and applicability of text editors have evolved to the point that a reassessment of the role of the text editor is now appropriate. This reassessment is particularly needed in the light of other, unrelated evolutions of technology, namely the utility-like interactive service now feasible with networking technology and the radical increases made in available computing power for a given investment. 12a2

There are many different attributes by which text editors may be evaluated. At a high level, the following list is suggested: 12a3

1. The computing power available. This attribute covers a number of characteristics, including the number and scope of the commands available, the number of diverse tasks the system may perform, including text entry, storage, retrieval, editing, formatting, printing, typesetting, graphics, composition, and others. The range of text applications is also relevant, as the requirements for an effective system for program creation and manipulation are significantly different from those for a system designed to support the publication of formal mathematical papers.

2. The human factors relating to use of the editor. This attribute includes the sophistication and training required for effective use of the editor, convenience of use, the expressive power built into the command language, the input error tolerance provided, and the difficulty of adapting the editor to a specific application. Human factors involves not only the design of the editor software, but also the hardware and protocols involved in the user-system dialogue.

3. The resources necessary to support the system. This is both a hardware and a human requirement, as often tradeoffs may be made between human effort and hardware resources. The human effort required may be that of the user, or may additionally include the staff of support personnel necessary to support a large hardware installation.

These attributes are clearly not orthogonal, and may not be entirely complete. They are an attempt to isolate the factors to be considered in the selection of text editors. 12a4

As noted in attribute (3) above, there are tradeoffs between resources. The more generally correct statement is that there are tradeoffs to be made between all three attributes. A generally famous law of biological specialization states that the more suited an animal is for a particular task the less adaptable it is to other tasks, and conversely, the more generally adaptable an animal is the less well suited it is to any specific task. Applied by direct generalization to computing, and to editors in particular, it is clear that any completely general purpose system will either be less effective than a specialized editor, or else will require additional computing resources to perform the task equally well. 12a5

The common denominator is, of course, money. For enough money the resources can be acquired to permit the use of the more general system, or the handicaps imposed by the specialized system may be overcome. The interesting problem, and the one facing the NSW, is to provide the best possible editor facilities at reasonable cost. To do so it is required to examine the applications to be supported by the NSW editors, and to determine the degree of specialization and generality required. Based on such an analysis, one may then determine if a single generalized system is required, or if a combination of systems, some general, some specific, is better. 12a6

APPLICATIONS 12b

There are five applications of text editing systems we will discuss here. These applications are: 12b1

1. Document preparation.
2. Program development.
3. Configuration Management.
4. Communications (Electronic Mail).
5. Information Storage and Retrieval.

Document preparation conventionally involves at least three people, namely the author, the typist, and the artist. When done with the aid of a text editor some or all of these personnel may be required to interact with the text editing system. Factors important in the evaluation of a specific text editor for document preparation include the simplicity of use of the system, the size of the document to be prepared, the security classification of the document, the updating frequency, and the volume of the document revised by each update.

If a facility does not presently utilize a text editor the cost to adapt to one for a specific project may be too high unless size or updating factors improve the economics of use. If a text editor with

insufficient power is selected then it becomes difficult to prepare the document. The specific features necessary may include size of text which may be handled, editing capabilities, output formatting capabilities, graphics, and configuration management. The range of power for such systems spans the spectrum from simple magnetic card typewriters, such as the IBM MTST system, through editors such as the IBM ATS or TRW MPS manuscript systems, to extensible large systems such as NLS.

Program development (ignoring for now the issues of requirements definition, design, traceability, etc.) involves the activities of coding, debug and test, and modification. This task is typically performed by programmers, who often are found to prefer a text editor with a cryptic human interface, due to the minimum amount of redundant interaction required to perform a task. Because programs are often quite large, the size of text which may be handled is important, as is the range of available commands (again, to minimize the redundant interaction required) and the availability of some form of configuration control.

Configuration control may be as primitive as simply keeping a backup version of the changed file, or may include more sophisticated capabilities such as recording the identity of the programmer making changes, providing access control to prevent unauthorized changes, and the provision of some form of change audit trail. Text editors for programming support have been developed which are intelligent about the structure of the programming language and thus provide clerical checks for the programmer. Systems of this sort include work at ISI, the INTERLISP editor, and various commercial COBOL editors. The resources required are critical here because programmers as a group are particularly unwilling to tolerate poor response time unless no alternative exists. The manifestation of this fact in the context of the NSW may be that the more powerful editors which do not offer some minimum response time will generally be rejected by programmers in favor of less capable systems offering acceptable performance.

Configuration management has already been mentioned in the two application areas above, and indeed pervades all uses of textual material. The relevance of text editors in the case of configuration management is that they provide the opportunity to accomplish such functions automatically while the textual manipulation is being carried out. Examples of such services include the statement signatures provided by NLS and the comprehensive version/revision/access control mechanism of the TRW SDVS system.

The use of text editors for electronic mail over networks is usually restricted to very long messages or to those messages which are to be in a "clean" format. Editors also provide capabilities for viewing messages, but again this is often used only for long messages. Most

mail systems incorporate primitive text editors, which accounts for the limited use of separate editors. Conversely a number of editors (e.g. XED, TECO and NLS) are either integrated with message systems or incorporate their own such systems. That system which provides the most sophisticated service normally required, and which does not require a cumbersome or time-consuming interface, will normally be found in use.

Finally, text editors provide the capability to collect and organize data for later retrieval. This capability is used, for example, to provide reference and abstract catalogues such as the well known set of chemical abstracts. The data bases for globally used information collections are often extremely large, and so data base query/update techniques are used rather than normal text editors. Private collections of information are within the scope of text editors, for example the accumulation of a bibliography while research is in progress. For this application, retrieval power and ease of use, as well as response time, are of more importance than sophistication and power.

There appear to be two motivations for using computer-driven text handling systems: capability, and productivity. In the first case, a computer system may make possible tasks which are simply too demanding to be performed manually. The chemical abstracts are good examples of this -- it is worth noting that in almost no cases do the text editors we are discussing here, however, provide capability unavailable by other means. 12b2

Instead, the motivations for using text editors appear to be almost exclusively oriented around productivity. A programmer could, for example, maintain his programs on cards in his desk drawer (as many unfortunately do); documents are often, if not usually, prepared by manual methods using conventional typewriters, and configuration management may be accomplished manually giving rise to a enormous volume of paperwork. Similarly, the U.S. Post Office has provided mail service for some time now, as do most companies internally, and may be expected to do so for some time to come. The advantage of the text editor to the programmer is that modifications are easier and the clumsiness of cards and card readers is avoided; the advantage to the documentor is that editing is often easier and faster (especially important for documents updated frequently); the advantage to the configuration manager that much routine paperwork may be subsumed by automated processes, and so on. 12b3

Clearly, then, text editors must justify their usage on economic grounds, not only in the context of which text editor to use, as has been the subject of many recent surveys and comparisons, but also in the context of whether or not a computer-driven system should be employed at all. The economic gain in productivity that will be measured for a given application must be weighed against the costs in hardware and personnel, and the time frame over which the system must justify itself must be defined. 12b4

In certain cases the choice will be easy, such as the case of a

documentation group which processes mainly classified work. In this case, as the net is not cleared for classified access, the only alternatives are to procure a local machine to be used with the editor system in a stand-alone mode, or to rely on conventional techniques. If the investment in hardware is on the order of $500K or more (as is the case with the machine required to host the NLS system), the acquisition of a machine is difficult to justify without extremely large volume. Alternatively, if the work is unclassified, then the NSW could fulfill its role of tool test stand and the implied decision about the level of technology required could be made after trying all of the editors available on the NSW, or at least those which have an apparent application. 12b5

The above discussion shows that the importance of the attributes defined in the introduction will vary relative to one another depending on the application. Some applications will be found to require the same person to operate in more than one of the five roles at different times during the process. 12b6

The requirement for command power depends on the sophistication of the user as well as the application. For instance, commands may be made very limited and specific for editors used by unsophisticated programers; such editors are commonly found in BASIC systems and have limited capability but high comprehensibility and ease of learning. Conversely, although the sophisticated programmer may desire what seems to be an unusually wide range of commands, it appears that sophisticated documentation applications may require the most power including: the ability to set type fonts and formats, do graphics and indexing, and other generally non-trivial functions. 12b7

The human interface primarily consists of the dialogue language used between editor and user, the human factors engineering (such as special terminal design or utilization), and the"robustness" of the system (i.e. do natural-seeming commands in fact produce the expected result). Small editors often employ extremely cryptic command languages and make no use of special features. Systems with the size and history of NLS or XED, however, often incorporate simple yet powerful command languages and are able to utilize special terminal features, such as the lineprocessor terminal for NLS or the HP terminal for XED, to improve the capability and facility offered to the user. The nature of the interface language is very application critical -- programers, as hypothesized earlier, may be willing to put up with excessively terse input, whereas managers attempting to query a data base may not utilize the system often enough to accomplish sufficient memory retention of the commands. The result is that the system may well not be cost effective for them to use. 12b8

The resources consumed by a text editor cannot be measured solely in terms of the computer cycles and hardware necessary to perform a benchmark series of tasks, as is assumed in a number of the references. The editor is a component of the total system involved in accomplishing the benchmark, a

system which includes the computer resources, the concomitant facilities to house the computer, the operations staff, if any, and the user. Meaningful measurement of the cost of operation for a text editor, then, must measure the total cost of achieving a result. 12b9

The determination of resource usage, therefore, is a multi-dimensional task. Included will be a need for computer hardware or software monitors and some means to measure the human resources. The meaningful measurement of human resources is complicated by the wide range of human productivity among individuals and over a period of time. 12b10

This is to say that the computation of potential cost-effectiveness is not a simple task. While it is easy for General Motors to determine which of several competing lathes is more cost-effective over a relatively short period of time, the more complex nature of textual tasks leads one to evaluations of additional benefit, and the effects of human variation in productivity require a longer period of measure. 12b11

RECOMMENDATIONS 12c

The choice of action for NSW to take with respect to text editor installation is determined by economic information not available at this time, namely the costs involved with installing, maintenance, personnel training and computer resource consumption of an "average" editor on the NSW. Given that these costs are small enough, then there is no reason to require that NSW users standardize on one editor versus another. In such a situation the proliferation of editors now prevalent on the ARPANET would be expected to transition to the NSW as use of NSW grows, and the selection of one editor over another would remain a personal decision as it is now. 12c1

Conversely, if the total cost of editor installation and maintenance is high, then resource limitations will dictate that some selection of editors to be installed must be made. In this case, the following steps should be undertaken to determine the proper choice of editors. (The expected diversity of applications would seem to imply that more than one editor will be installed.) 12c2

1. Determine the application mix of the NSW and the workload for each application type. The segmentation of workload types should be made according to the requirements placed on the text editor system as discussed above. Establish the importance (weight) of each of the five categories of editor use.

2. Rank the available editors in terms of applicability to the weighted workload mix defined in step one. Two types of ranking could be used: one wherein the rank for an editor is some average of its value in each application, and the other wherein its rank is the maximum for any of the five applications. In the latter case the rank must also include a description of the corresponding application. Intuitively, the latter

method would seem to offer better results: the capability offered to a variety of applications is better served (in the limiting case) by one editor specifically tailored for each application rather than by a multitude of editors each attempting to satisfy all applications.

3. Estimate the total cost for each of the editors under consideration.

4. Determine the covering which best satisfies the application workload requirements for the least cost. Of all the possible coverings (i.e. combinations of editors to be made available), a partial ordering will occur; one ordering will provide more benefit at more cost relative to another. The means for decision between such alternatives is not clear; our only recommendation in this event is that past experience with the editors must be combined with an understanding of the available resources in the decision making process. In any case, if the decision is so close as to be truly unclear, then it may well be the case that the particular decision made is in fact irrelevant, and that either alternative would serve equally well.

REFERENCES and BIBLIOGRAPHY 12d

Bair, J. H., The Capabilities of the Augmented Human Intellect On-Line System Compared with Text Editing Systems: A Case in Point, Stanford Research Institute, Journal Accession No. (364??,), August 1976. 12d1

BBN, Inc., TENEX TECO Text Editor and Corrector Manual, Bolt Beranek and Newman, Inc., October 1973. 12d2

Benjamin, A. J., An Extensible Editor for a Small Machine With Disk Storage, Communications. ACM 15, 8 (August 1972), 742-747. 12d3

Berns, G. M., Description of FORMAT, a Text-Processing Program, Communications. ACM 12, 3 (March 1969), 141-146. 12d4

Berns, G. M., The FORMAT Program, IEEE Transactions on Engineering Writing and Speech, Vol. EWS-11, No. 2 (August 1968), 85-91. 12d5

Bourne, S. R., A Design for a Text Editor, Software Practice and Experience 1, (1971), 73-81. 12d6

Bullen, R. H. and J. K. Millen, Microtext - The design of a microprogramed finite state search machine for full-text retrieval, Proceedings Fall Joint Computer Conference, (1972), 479-488. 12d7

Crain, L. A., PCS/TEXT vs NLS, Air Force Data Systems Design Center, Journal Accession No. (27393,), January 1976. 12d8

Flowerdew, S. J., LEFT: A Language for Editing and Formatting Text,

University of Illinois Department of Computer Science Report No. 291, (October 1968). 12d9

Glantz, R. S., SHOEBOX - A personal file handling system for textual data, Proceedings Fall Joint Computer Conference, (1970), 535-545. 12d10

Hunter, L. W., A Data Representation Code for Text Processing Systems, International Journal of Computer and Information Sciences 1, 1 (1972), 29-42. 12d11

IBM Corp., IBM System/360 TEXT360 Introduction and Reference Manual, Form C35-0002. 12d12

Irons, E. T. and F. M. Djorup, A CRT Editing System, Communications. ACM 15, 1 (January 1972), 16-20. 12d13

Kaiman, A., Computer-Aided Publications Editor, IEEE Transactions on Engineering Writing and Speech Vol. EWS-11, 2 (August 1968), 65-75. 12d14

Lee, S. R., et al, The Economics of Text-Editing Functions: Cost Effectiveness Analysis of NLS and Other Systems, Stanford Research Institute, Journal Accession No. (16264,), May 1973. 12d15

Madnick, S. E. and A. Moulton, SCRIPT, An On-Line Manuscript Processing System, IEEE Transactions on Engineering Writing and Speech Vol EWS-11, 2 (August 1968), 92-100. 12d16

Moore, C. G. and Mann, R. P., CypherText: An extensible composing and typesetting language, Proceedings Fall Joint Computer Conference, (1970), 555-561. 12d17

Moudry, J., A Notation Describing Corrections in Files, Software Practice and Experience 2, (1972), 279-285. 12d18

Poole, M. D., Implementation of an Editing Algorithm Allowing Repeating Corrections, Software Practice and Experience 1, (1971), 373-381. 12d19

Rich, R. P. and A. G. Stone, Method for Hyphenating at the End of a Printed Line, Communications. ACM 8, 7 (July 1965), 444-445. 12d20

Riddle, E. A., A Comparative Study of Various Text Editors and Formatting Systems: Annotated Draft, Air Force Data Services Center, Journal Accession No. (26261,), August 1975. 12d21

Stanford University, WYLBUR Reference Manual, Stanford University Computation Center Systems Documentation Library, (March 1970). 12d22

Tepperman, A. and J. Katzenelson, A Format Editor, Software Practice and Experience 2, (1972), 219-230. 12d23

USC/ISI, XED User's Manual Beginning Instruction, University of Southern California Information Sciences Institute, No. ISI/TM-76-3, May 1976. 12d24

Van Dam, A. and D. E. Rice, On-Line Text Editing: A Survey, Computing Surveys 3, 3 (September 1971), 93-113. 12d25

Yonke, M. D., A Knowledgeable, Language-Independent System for Program Construction and Modification, University of Southern California Information Sciences Institute No. ISI/RR-75-42, October 1975. 12d26

APPENDIX E - NSW TOOLS CONFIGURATION AND QUALITY MANAGEMENT PLAN 13

TABLE of CONTENTS 13a

page 13a1

ILLUSTRATIONS

13a8

page

1.0 OVERVIEW 13b

The National Software Works is to provide a repository of software tools, and to provide for user access to these tools through a common communication network. As the user interface for these tools, the National Software Works will be susceptible to the problems inherent in acquiring and maintaining software in general, and tools in particular. The purpose of this report is to define a Tools configuration and Quality Management Plan for the National Software Works to deal with these problems. 13b1

There are several significant events which must be controlled by the plan: The acceptance into the National Software Works of a new tool; the controlled maintenance of the tools in response to user identified problem reports; the commissioning of new tools; and the directed augmentation of a tool. This plan was designed to control both the NSW-resident software and related documentation for these events. It indicates the intaractions that must take place to provide a controlled software laboratory environment and to manage the content of the NSW resident tools. 13b2

1.1 SCOPE 13b3

The tool configuration and quality management procedures cover the activities required to identify, establish, and control the design and content of software packages that are released to and maintained by NSW. Included are the procedures for accepting the tools and their relevant documentation, controlling design changes, processing discrepancy reports, and managing the overall control process by the Tools Review Board (TRB). Design change control procedures do not apply to entire computer software packages while they are being designed, coded, and tested by the tool developer.

1.2 APPLICABILITY 13b4

These procedures apply to the members of the NSW in the following roles:

- o tool developers, submitting tools for inclusion into the NSW;
- o tool users, identifying problems in the tools in the NSW;
- o the Tools Review Board, responsible for maintaining a stable user interface for tools in the NSW;
- o those directed by the TRB to install and modify the tools in NSW and to provide relevant documentation to users.

2.0 APPROACH 13c

The NSW will contain numerous software tools developed by different organizations. Configuration and Quality Control are normally designed for individual software products; it is not practical for the NSW to establish a separate control procedure for each of the tools which are to reside in

NSW. Therefore, as a matter of practical convenience, all NSW-resident software packages are to be controlled by this one single plan. 13c1

This plan assumes that the development and maintenance of many software tools in NSW is funded by other organizations, i.e. that NSW has no direct control over the development and maintenance prior to being installed in NSW. However, certain standards must be met and maintained if tools are to be provided to NSW users. It is necessary that the configuration of an evolving tool, at any point in its useful life, be recorded and identifiable so that past configurations can be re-established. In the interest of economy of procedure, design changes originated by the tool developer are processed with a minimum of procedural control. The procedure for incorporating design updates contained here require only that the users of the affected software be informed of the change before it is installed. In addition, in some circumstances NSW will play an active role in acquiring and augmenting tools. This plan addresses all of these situations. 13c2

2.1 PARTICIPATING ORGANIZATIONS 13c3

Responsibility for the performance of configuration and quality control procedures is shared by the following organizations:

NSW Project Office -- responsible for overall management of the NSW Project, including management of the NSW system and all resources available through it.

Tools Review Board -- responsible for managing the acquisition, acceptance and modification of NSW tools.

Configuration Management Office (CMO) -- responsible for installation and maintenance of the physical integrity of all NSW resident software tools. All configuration control forms are submitted to this office for processing and forwarding for action by participating organizations. This office provides the clerical and analysis support for configuration control activities. The CMO acts as the central point in the NSW for receiving all deliverable software package material and change material and for handling all configuration control transactions.

NSW Member Organizations -- responsible for submitting the software packages for entrance into NSW, for installing the packages on their respective computers, and for installing modified packages under the direction of the CMO.

2.2 EXPLANATION OF TERMS 13c4

Definitions for several acronyms and terms used frequently throughout this document are provided below:

NSW Transmittal Memorandum (NTM)

This form is the standard vehicle for: Submitting material for new software packages, changes to existing software packages or modules, new and revised documents, and software correction decks; and provides explanations of Discrepancy Report Forms (DRF) and engineering change proposals (ECP). The NSW Transmittal Memorandum may be originated by any NSW associate contractor participating in NSW software development and/or maintenance. An NTM accompanies a software package, a new version of a software package, or a set of changes. A description of the contents, identification of incorporated changes, installation instructions, and/or a description of possible problems or known errors should be included. (See Figure 2-1 at the end of this report.)

Baseline

A baseline may be all or part of a software package and its related documentation, designated at some point as the base upon which to make all future approved and controlled changes that affect software operation, design, or equipment interface. At the designated point in time, the identified material is subject to these procedures. Approved design requirements specifications may be baselined while software is being developed and is referred to as a requirements baseline. The entire software package, after it has been accepted by the NSW Project Office, is referred to as the product baseline.

Engineering Change Proposal (ECP)

This is a standard NSW form used by NSW members for proposing design changes to software packages (in response to an approved Design Change Request) for which the proposing contractor has design development or maintenance responsibility. It is used to identify total impact, schedules, and resources required to accomplish a change in detail sufficient to provide a basis for TRB Action. This form may be used in the processing of any change to a software package, but it is required for directed changes. (See Figure 2-2 at the end of this report).

Change Control

Change Control is the process in configuration control that permits design changes to be made to a baselined software package.

Configuration

This term denotes a collection of software modules which may be process constructable or otherwise assembled to form an

independently functioning deliverable computer program or application. It is the primary software entity which is managed by configuration control procedures. A configuration made up of specifically identified (by version) modules is called a configuration version.

Configuration Index

The configuration index is a file maintained by the Configuration Management Office (CMO) for each software package. It contains the current listing of basic document issues and updates of specifications and supporting documents, together with identification of all changes incorporated in the documentation and/or computer software.

Configuration Version (number)

This term applies to a software configuration that is composed of specifically identified (by version) code modules. A configuration version may have some of its constituent modules existing in more than one version (see Configuration). This is the right-most two digits in the software package name (SPN) and is called the version designator.

Configuration Version Variation (number/letter)

This term is used to designate any of the several variations, composed of a particular set of module versions selected from a configuration version, which can be constructed and executed.

Corrections Log (CL)

A permanent file of all corrections, discrepancies or problems reported for every tool known to the NSW.

Design Change Request (DCR)

This is a standard NSW form which, when executed by an NSW member, provides the vehicle for bringing a suggested software package change to the attention of the TRB and the CMO. This form is used to initiate a directed change. It is not necessary to use the form for those design updates for which TRB coordination is not required (See Figure 2-3, at the end of this report).

Directed Design Change

This term denotes a design change to a software package that is coordinated by the CMO. Its disposition is determined by TRB.

Discrepancy Report Form (DRF)

This is the standard NSW form on which an NSW user documents a software discrepancy. The form is used when it appears necessary to advise other users of a particular software package that a discrepancy exists or when a record of a software discrepancy is needed. (See Figure 2-4, at the end of this report.)

Module Version (number)

This term refers to a particular code module having unique identity and differing from other modules performing like functions because of the implementation of changes and/or corrections, or because of a different functional approach.

Software Package (SP)

For a particular software application, the software and related documentation compose the deliverable package which is subject to control by the NSW configuration control procedures (see Configuration) after acceptance into the NSW.

Software Package Name (SPN)

This term is assigned by the Configuration Management Office (CMO) in the NSW to a group of computer software items associated with a particular software package that is controlled for use in the NSW.

3.0 ACCEPTANCE OF A TOOL INTO NSW 13d

Entrance of a tool into the NSW implies a certification of a minimum standard of quality, and implies a commitment of a level of effort by the NSW Tools Review Board and the CMO to maintain a stable user interface. Thus entry into the NSW is controlled by the Tools Review Board. Figure 3-1 provides an overview of the steps necessary to install a software tool in the NSW. 13d1

3.1 Request Inclusion of Tool into NSW 13d2

The tool producer, or the organization who will be responsible for maintenance of the tool, will request entry of the tool into the NSW. Entrance into the NSW implies a commitment on the part of the organization to maintain the tool in accordance with CMO direction. The request is made to CMO, which will classify the tool, compare it to those in the current inventory, investigate the status of the documentation, and make a recommendation to the Tools Review Board. Upon approval of the request, the TRB will direct the CMO to perform the acceptance testing and configuration analysis of the tool.

3.2 Release Documentation and Support Material 13d3

On the date scheduled, the producer releases the software packages materials and all relevant documentation to the CMO for acceptance testing and configuration audit. This release is accompanied by an NSW Transmittal Memorandum. A copy of the program, user's manuals, and other supporting documentation as directed by the Tools Review Board will comprise the minimum acceptable documentation.

3.3 Log In and Process Materials 13d4

The CMO receives and logs in the software materials delivered by the tool producer. All documents contained in the release are cataloged for inclusion into the NSW Library. The CMO examines the material received, and designates an SPN for the software package, if an SPN has not been already assigned. In addition, abstracts and announcements are prepared to notify NSW users. The CMO also authorizes the necessary filespace for its retention and use.

3.4 Performance of Configuration Audit and Analysis 13d5

The CMO coordinates as necessary with the tool producer for running acceptance tests to meet the standards established by the TRB. The tool is used by members of the CMO in a variety of situations to establish the consistency of the software, its documentation, and its utility. The results of this analysis are included in a Test and Evaluation Report. The results of the test, and the accounting of all relevant materials, are included in the configuration audit. Any discrepancies noted during the acceptance tests are forwarded to the producer using discrepancy reports, and these are included in the configuration audit also.

3.5 Tool Acceptance 13d6

On the basis of the Test and Evaluation Report, the Tools Review Board may accept the tool into the NSW by directing the CMO to install it as an NSW resource. Allocation, control and use of the tool will then be under the direct control of the CMO. The CMO notifies users of the availability of the tool, and upon request provides users entitled to access the tool with appropriate NSW rights and documentation.

The TRB may provide approval contingent upon the correction of known discrepancies. Subsequent corrections are provided to the CMO on NTMs, both of documentation and program code. These corrections are processed in accordance with the procedures of the next section.

3.6 Options 13d7

There are several options to be considered in the above procedure.

The standards to be met by tools being delivered to NSW by producers must be defined. These can range from "minimal" to a complete configuration package of required documentation; or various categories can be established, each with their own standards. The degree of testing is yet another variable to be established by the NSW TRB.

The role of the CMO in acceptance testing the proposed tools is a significant variable. This testing can range from a cursory inspection and submittal of a few sample runs, to a complete software acceptance testing.

There may be several categories or classes of tools for which different levels of tool certification may exist. These may range from experimental or prototype tools useful to the Research and Development community to production quality tools which are used to support large, operational computer-based systems.

4.0 TOOL MAINTENANCE CONFIGURATION CONTROL 13e

The purpose of tool maintenance configuration control is to provide the user a stable interface to tools in the NSW, yet provide for tool modification and adaptation to correct known deficiencies and introduce new capabilities. Several functions are necessary to accomplish this: 13e1

- o collection and control of user-submitted discrepancy reports
- o controlled modifications to correct deficiencies
- o verification and certification of tool modifications
- o retention and control of tool configurations
- o user notification of tool status
- o use audit of tool configurations

Figure 4-1 presents an overview of the Discrepancy Reporting Procedures. Discrepancies that are detected in a tool resident on the NSW are reported by this procedure so that they may be made known to all users of the affected tool, and so that prompt remedial action can be taken to correct the discrepancy. 13e2

4.1 REPORTING SOFTWARE DISCREPANCIES 13e3

Users of NSW resident tools prepare and submit a Discrepancy Report Form (DRF) whenever they encounter a malfunction in the use of a tool and/or error in the documentation. The originator of the DRF submits the form to the CMO.

4.2 INITIAL PROCESSING OF A DISCREPANCY REPORT FORM 13e4

Upon receipt of a Discrepancy Report Form (DRF), the CMO assigns it a control number and logs it into the Corrections Log of the tool to which

it pertains. A copy of the DRF is made available to other users of the tool in order to alert them to the discrepancy (this could be done by having a Corrections Log associated with each tool, and a special notification to the user when he logs on for the tool to flag new discrepancy reports). A copy of the DRF is forwarded to the NSW member responsible for maintaining the tool for action. The CMO keeps a status record in a Corrections Log for all DRFs by tool so that progress of correcting a DRF may be known at any time.

4.3 REVIEW FOR DESIGN IMPACT 13e5

The maintenance contractor first examines the reported discrepancy for impact on the design specifications of the software package. If the DRF has no impact, work to correct the discrepancy begins. If the DRF describes a design change rather than a discrepancy, the DRF is cancelled in the Correction Log and the CMO notifies the DRF originator of that action.

4.4 CORRECTION PREPARATION 13e6

If so directed by the CMO, the maintenance contractor prepares a correction of the reported discrepancy. After checking out the validity of the correction, the maintenance contractor prepares an NTM to submit with the correction. The correction and the NTM are submitted to the CMO for incorporation into the affected software package.

4.5 INSTALLATION OF THE CORRECTION 13e7

The CMO receives the correction material identified by an accompanying NTM and notes receipt of the correction in the Corrections Log. The CMO has any material cataloged and/or filed that is to be deposited in the NSW Library which is relative to the correction. This includes change pages for document error corrections.

The CMO alerts the Analysis Group to the need for installing and testing a correction deck. If the correction is successfully tested, the Analysis Group advises the CMO to release the corrected file. This is accomplished by the CMO authorizing the appropriate NSW member to install the tool. The Version Designator in the SPN is modified to indicate the correction, and the NSW Library is updated to reflect the existence of the new version of the software package. Following the release of the corrected file, the CMO closes the outstanding DRF by noting it as corrected in the Correction Log for the software package.

The CMO also advises the Analysis Group when document correction materials (change pages) are received. The Analysis Group reviews the document corrections to determine their validity.

If a correction does not test properly, or if a document correction is

in error, the CMO notifies the maintenance contractor and holds the DRF active until a workable correction is received. The originator of the DRF and other users of the affected software package are informed when the reported discrepancy is corrected.

The updating of the NSW Library includes the updating of the NSW Tool Summary available to NSW users. This summary should contain such information as:

- o listing of open Discrepancy reports
- o listing of correction and release reports
- o Tool Abstract Master List of all tools programs in program ID order with its associated text and all keywords describing the program
- o listing of all tools programs by program language or computer configuration
- o listing of all tools programs in keyword alphabetical order
- o listing of all or selected tools programs ordered by functions
- o listing of closed problem reports since last update
- o listing of NSW Transmittal Memoranda.

This information should be contained in a data retrieval system to provide easy user reference. The CMO is responsible for keeping this information current.

5.0 DESIGN CHANGE PROCEDURES 13f

There are two types of design changes which might occur to a tool in the NSW; a design update, originating with the tool producer or maintainer to make available additional capabilities not found in earlier versions, or a negotiated design change. The latter term applies to design changes requested by NSW users other than the NSW member responsible for tool maintenance. An important function of the NSW is the coordination of desired modifications to the standard NSW tools between users, funding sources and producers. 13f1

We assume that the NSW will sponsor an NSW User's Group, which includes special interest groups in various types of tools (e.g., program management tools, engineering and scientific tools, design tools, development tools, test tools, V&V tools, configuration management tools). These Special Interest Groups (SIGs) should play a role in tool modification as well as tools acquisition. 13f2

5.1 DESIGN UPDATES 13f3

Design updates do not require TRB approval. However, should the tool developer question whether he should initiate a particular design update, he may submit a Design Change Request for the contemplated change for discussion with the appropriate SIG (see Section 5.2).

5.1.1 Originating a Design Update

Design updates originate with the NSW member currently responsible for the tool to be changed. The changes are identified, initiated, and the relevant documentation modified to achieve a consistent software and documentation package. The software is installed on the same NSW machine as the currently residing tool, and the configuration package is forwarded to the CMO with an NTM.

5.1.2 Receive and Process Material

The CMO received from the NSW member all the materials needed to update documentation to install the design change in the NSW. An NSW Transmittal Memorandum must accompany this material. The CMO has any material relative to the design change cataloged and/or filed, and deposited in the NSW Library. The CMO makes an entry on the Change Log for the tool to show that a design change has been received. The entry is made under the Software Package Name (SPN) for that tool. The CMO informs the Analysis Group that the change material is ready for review and/or integration into the software package which it changes, and that acceptance tests can be conducted if the software is modified.

5.1.3 Change Integration and Installation

The CMO Analysis Group tests the modified software package to insure that the change works as intended, and reviews the relevant modified documentation. If the test results indicate that the change is successful, and the documentation review for accuracy is found to be satisfactory, the CMO notifies the tool producer of acceptance.

5.1.4 Release of Design Update

When the CMO has successfully tested the design update, the CMO updates and releases the version for NSW users. If the design update is incompatible with the existing version of the tool, both versions may have to be maintained for a considerable period of time; even when the use of the old version falls to zero, that version should be archived to assure tools continuity by infrequent users of the NSW. To install the modified version, the same activities should take place as for the updates described in Section 4.5 for discrepancy corrections.

Design updates do not require coordination action by the Tool Review Board in order to initiate and implement a design change in an NSW resident tool. Parties to this transaction are the tool producer and the CMO. Figure 5-1 presents an overview of the activities involved.

5.2 NEGOTIATED DESIGN CHANGES

A Negotiated Design Change is one requested by an NSW user who is not the tool producer. An important function of the NSW is the coordination of modifications made to tools and documentation. It is assumed that the NSW Users Group (and constituent Special Interest Groups for each tool category) will provide forums for informal discussions of proposed tool changes, and that the Tool Review Board will serve to formally coordinate the tool modifications between users, potential funding sources, or developers. Thus, Negotiated Design Changes will require action from the TRB in order to initiate and implement a design change to a tool. An overview of the procedure appears in Figure 5-2.

5.2.1 Originating a Negotiated Design Change

All Negotiated Design Changes are initiated by a Design Change Request (DCR) form. This form may be originated by an NSW user, the TRB, or a tool developer. Only one change should be requested on a DCR. The originating organization submits a Design Change Request to the CMO for TRB action.

5.2.2 Receiving and Initial Processing of Design Change Requests

The CMO, upon receipt of a DCR, makes certain that all necessary information has been included. If the DCR is incomplete, the CMO contacts the originator and obtains missing information, a step which may require that the DCR be returned for completion. A properly completed DCR is assigned a control number from the particular software package series to which it belongs. This control number is used for status and record keeping purposes in the Change Log. Copies of the DCR are distributed by the CMO to those having an interest in the software package to which the DCR applies (the list is coordinated through the NSW Users Group), and to the TRB. The DCR copies are distributed prior to the TRB meeting at which the DCR is to be considered. The CMO places the DCR on the TRB meeting agenda.

5.2.3 Evaluating a Design Change Request

At the TRB meeting, the CMO and TRB evaluate each Design Change Request. Organizations having an interest in the tool to which a DCR applies are expected to express their position relative to the DCR through their representative at the meeting, or to the TRB by other appropriate means. Based on information present, the TRB acts to either return the DCR to the originator as rejected, or to assign it to the responsible contractor for a response.

5.2.4 Disposition of the Design Change Request

The DCR action taken is noted in the TRB minutes. These minutes, after approval by the TRB, are distributed to all recipients of the DCRs referenced in the minutes. The CMO makes sure that the assigned

responsible tool producers have enough information available to prepare an ECP.

5.2.5 Preparation of an Engineering Change Proposal

The tool producer, upon being notified of a TRB requirement for an Engineering Change Proposal, prepares an ECP based on the originating DCR and any additional information available from the requester. The completed ECP should include implementation information, as well as estimates of labor and computer time, so as to permit the TRB to coordinate the costs and schedule for the design change. The completed ECP is submitted to the CMO by the tool producer through his representative at the TRB.

5.2.6 Receipt of an Engineering Change Proposal

The CMO, upon receiving an ECP, ascertains that the data is as complete as possible. Any missing information is obtained from the responsible organization if possible. The CMO distributes the ECP to those organizations concerned with the tool; distribution is made in advance of the TRB meeting at which the item is to be considered. The CMO places the ECP item on the TRB meeting agenda.

5.2.7 Reviewing the Engineering Change Proposal

The ECP is reviewed by the TRB, with members having an interest in attendance at the TRB review meeting. Based on the information in the ECP, and on the advice received at the meeting, the ECP is approved or disapproved for implementation.

5.2.8 Acting on the Engineering Change Proposal

The CMO prepares and distributes the meeting minutes recording the action taken by the TRB after reviewing the ECP and related information. It should be noted that the TRB minutes will include action taken on both the DCRs and the ECPs noted here. ECP items disapproved are so noted in the minutes. For those approved, the CMO takes the necessary action to inform the responsible organizations that implementation of the ECP should proceed. For those DCRs and ECPs which are not approved by the TRB, the CMO shall notify the affected parties of the disapproval, together with any comments or suggestions dictated by the TRB. This notification shall be in the form of an NTM.

5.2.9 Design Change Implementation

The responsible organization receives ECP direction as indicated in the TRB meeting minutes. The organization then designs the change, and proceeds to implement it by updating, as required, the necessary

baselined material (code and documentation). After the organization has checked out the design change, he prepares an NTM to cover the change material that is submitted to the NSW.

5.2.10 Receipt of Directed Design Change Material

The CMO receives from the responsible contractor all the materials necessary to update documentation F{¬scc¬¬Cs;+¬sFC+¬¬55Y¬¬¬¬ relating to the design change cataloged and/or filed, if it is to be deposited in the NSW Library. The CMO makes an entry in the Change Log for the software package to show that a design change has been received. The entry is made under the SPN for that software package. The CMO Analysis Group informs the CMO that the change material is ready for review and/or integration into tne software package which it changes and that, if software is modified, acceptance tests can be conducted.

5.2.11 Change Integration and Installation

The CMO Analysis Group receives change material. Acceptance tests are conducted on the modified software package and, if test results indicate that the change is successful, the CMO analysis group informs the CMO. If the design change affects documentation, document changes are reviewed for accuracy; if they are satisfactory, the CMO is so informed.

5.2.12 Release of the Installed Directed Design Change

When the CMO Analysis Group informs the CMO that the design change has been tested successfully, the CMO updates and releases the computer library file containing the modified software package for use in the NSW. The Version Designator in the SPN is modified to indicate the run implementing the design change. The CMO makes an entry in the Change Log for the software package to show that the design change has been installed. The NTM is filed as supporting information. The CMO updates the Configuration Index for the software package to reference design change impact on software package documentation. The ECP is closed out by noting that event in the next TRB meeting minutes.

6.0 TOOL DEVELOPMENT 13g

An ongoing function of the NSW is the identification and coordination of development of new tools. The Tools Review Board should periodically evaluate the feasibility of developing new tools for the NSW, considering such factors as: 13g1

- o functions amenable to automation

- o potential tool complexity
- o availability and success of similar tools
- o universality of need
- o expected development or modification costs
- o potential labor and machine time benefits

Having determined the need for a tool, the TRB would then urge funding of such tools by members or governmental agencies. 13g2

In general, the configuration and quality control of the tools development effort will be dictated by the funding agency (or agencies). Proposals for such new tools should be coordinated in a fashion defined in Section 5.0 for technical content. 13g3

ILLUSTRATIONS 13h

NSW TRANSMITTAL MEMORANDUM Page 1 of_____ 13h1

TO: NSW CONFIGURATION CONTROL ORGANIZATION DATE REC'D

FROM:
Name Phone
Organization Date

SUBMITTAL OF:

[]Software Package Materials (itemize below)
[]Correction Deck
[]Other(explain)
[]Program/Module
[]Design Change Deck
[]Document/Revision

Name

Software Package Involved Document#

BRIEF DESCRIPTION OF TRANSMITTAL:

VERIFICATION: The Software Modules/Modification(s)

[]Have []Have not Been Tested At
By

Approved by Date

COMMENTS:

==========

(FOR NSW USE ONLY)

TRANSMITTAL ITEM(S):

Analyzed by: Date
Tested by: Date
Processed by: Date
Action Completed by: Date

SOFTWARE PACKAGE NAME APPLICABLE:

For additional forms, contact the NSW Project Office.

Figure 2-1: NSW TRANSMITTAL MEMORANDUM

ENGINEERING CHANGE PROPOSAL Page 1 of_____ 13h2

TO: NSW CONFIGURATION CONTROL ORGANIZATION DATE REC'D

FROM: NSW Number
Name Phone
Organization Date

DESIGN CHANGE FOR:
Program/Module Version
Software Package
Software Package Name (SPN)

DESIGN CHANGE TITLE (Function Changed):

DESCRIPTION:

JUSTIFICATION:

CONSEQUENCES IF NOT IMPLEMENTED:

DEVELOPMENT REQUIREMENTS:

OTHER MODULES/SYSTEMS AFFECTED:

EFFECT ON EQUIPMENT:

ALTERNATE SOLUTIONS:

SIZE OF DESIGN CHANGE (Estimated):

Small = Less than 1 manweek	Design/Coding	S[]	M[]	L[]
Medium = 1 to 4 manweeks	Testing	S[]	M[]	L[]
Large = Greater than 4	Documentation	S[]	M[]	L[]

Total Manweeks =
Total Computer Hours = Computer Type

ESTIMATED DELIVERY SCHEDULE:
PROPOSAL WORK APPROVED BY:

==========

(FOR NSW USE ONLY)

CMO ACTION Date
PROPOSED BY Date

For additional forms, contact the NSW Project Office.

Figure 2-2: CHANGE PROPOSAL

DESIGN CHANGE REQUEST Page 1 of_____ , 13h3

TO: NSW CONFIGURATION CONTROL ORGANIZATION DATE REC'D

FROM: NSW NUMBER

Name Phone
Organization Date

REQUEST DESIGN CHANGE IN:

Program/Module Version
Software Package Involved
Software Package Name (SPN)
Function to be modified

Responsible Organization(s)

Change Requested by

BRIEF DESCRIPTION OF CHANGE:

JUSTIFICATION FOR CHANGE:

CONSEQUENCES IF NOT IMPLEMENTED:

OTHER COMPUTER SYSTEMS OR MODULES INVOLVED:

INTERFACE OR EQUIPMENT PROBLEM INVOLVED:

ALTERNATE MEANS OF IMPLEMENTATION:

DOCUMENTATION AFFECTED:

Request Authorized by Date

==========

(FOR NSW USE ONLY)

CMO ACTION Date
PROCESSED BY Date

For additional forms, contact the NSW Project Office.

Figure 2-3: DESIGN CHANGE REQUEST

DISCREPANCY REPORT FORM Page 1 of_____ 13h4

TO: NSW CONFIGURATION CONTROL ORGANIZATION DATE REC'D

FROM:

Name Phone
Organization Date

PROBLEM WITH: (Indicate Applicable Areas)

[]Program/Module Version
Software Package Involved
Software Package Name(SPN)

[]Document(s)

[]Equipment

Responsible Organization(s):

Correction Requested by

DISCREPANCY DESCRIPTION:

ADDITIONAL DETAILS: (also, attach any available materials that show the existence of the problem.)

[]Attachments Included

==========

FOR NSW USE ONLY

DRF REVIEWED BY CMO Date
CORRECTIONS RECEIVED BY CMO Date
CORRECTIONS TESTED BY CMO Date
CORRECTIONS RELEASED BY CMO Date

For additional forms, contact the NSW Project Office

Figure 2-4: DISCREPANCY REPORT FORM

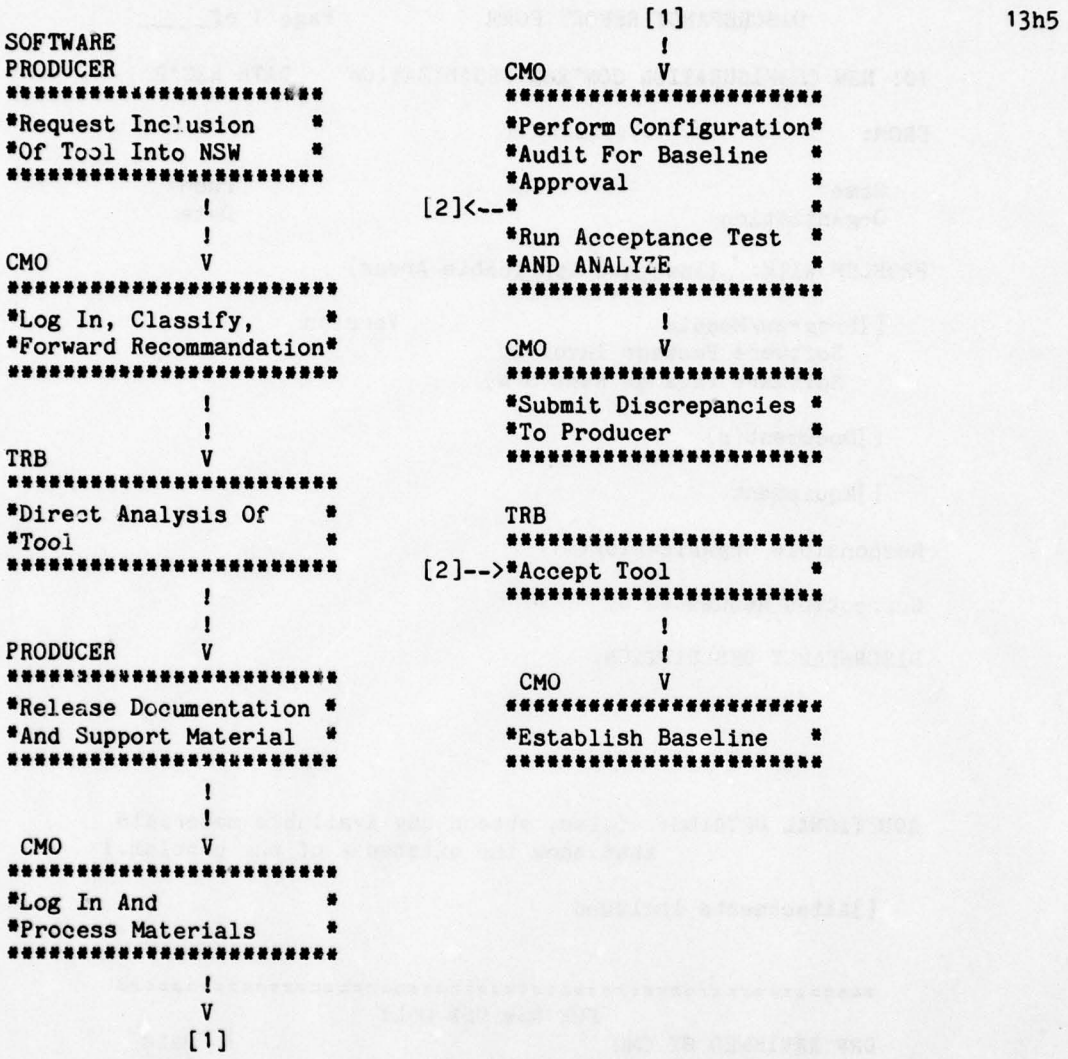


Figure 3-1: TOOL ACCEPTANCE INTO NSW

13h6

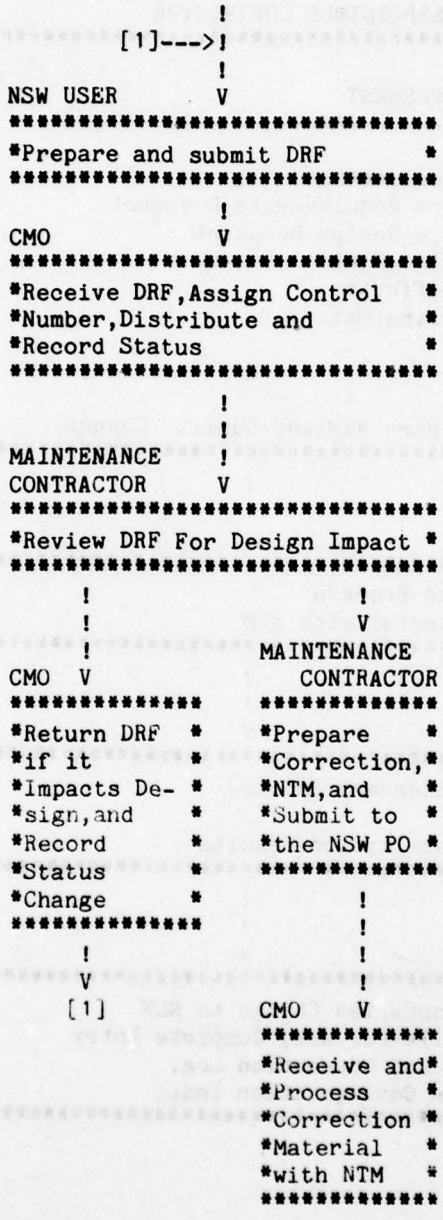


Figure 4-1: DISCREPANCY PROCESSING

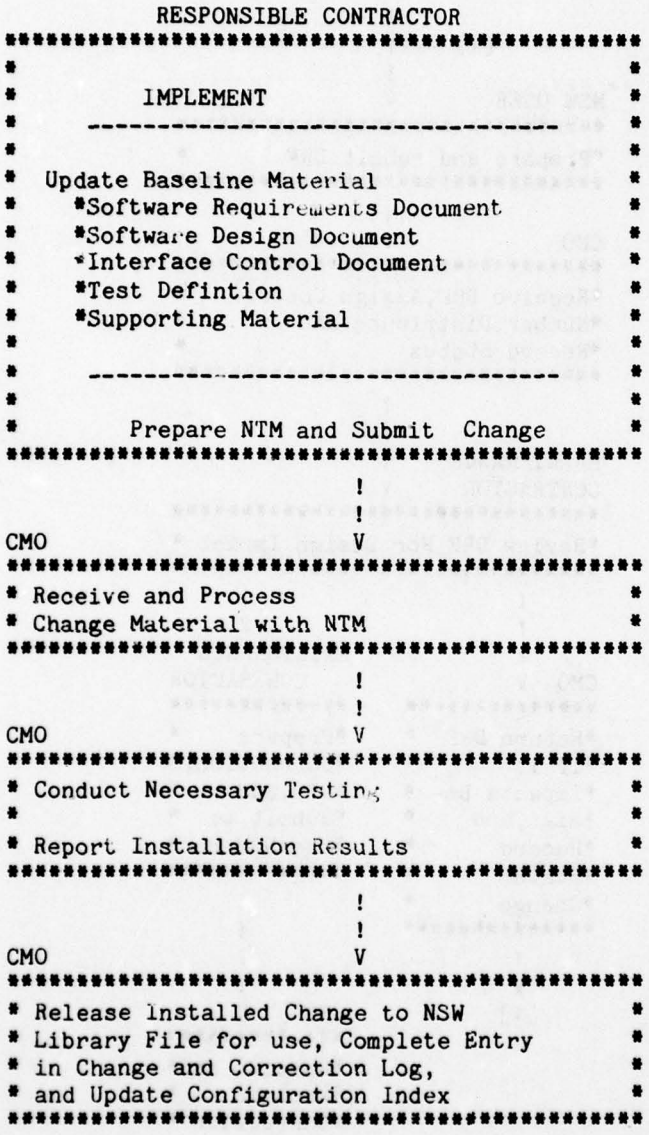


Figure 5-1: DESIGN UPDATE PROCESSING

AD-A052 996 TRW DEFENSE AND SPACE SYSTEMS GROUP REDONDO BEACH CALIF F/G 9/2
NSW FEASIBILITY STUDY.(U)
FEB 78

UNCLASSIFIED F30602-76-C-0270
RADC-TR-78-23 NL

2 OF 2
ADA
052996

END
DATE
FILMED
5-78
DDC

13h8

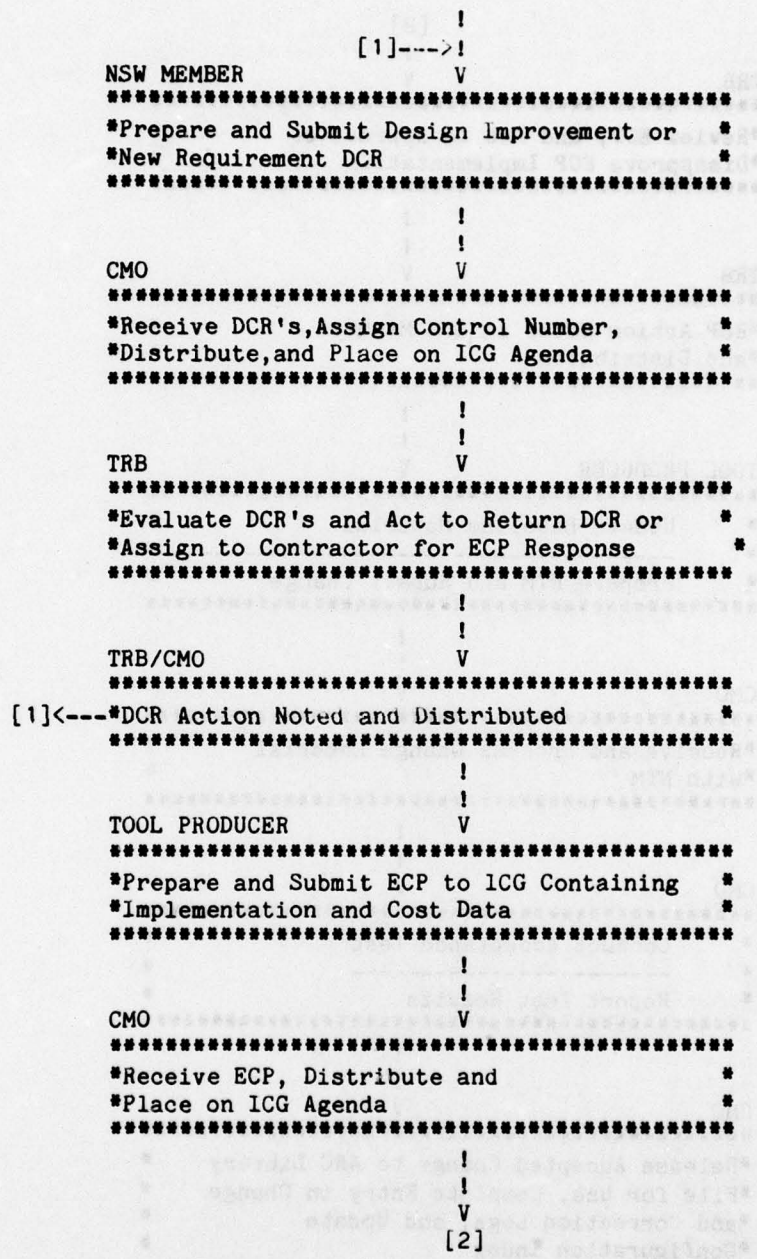


Figure 5-2: NEGOTIATED CHANGE PROCESSING

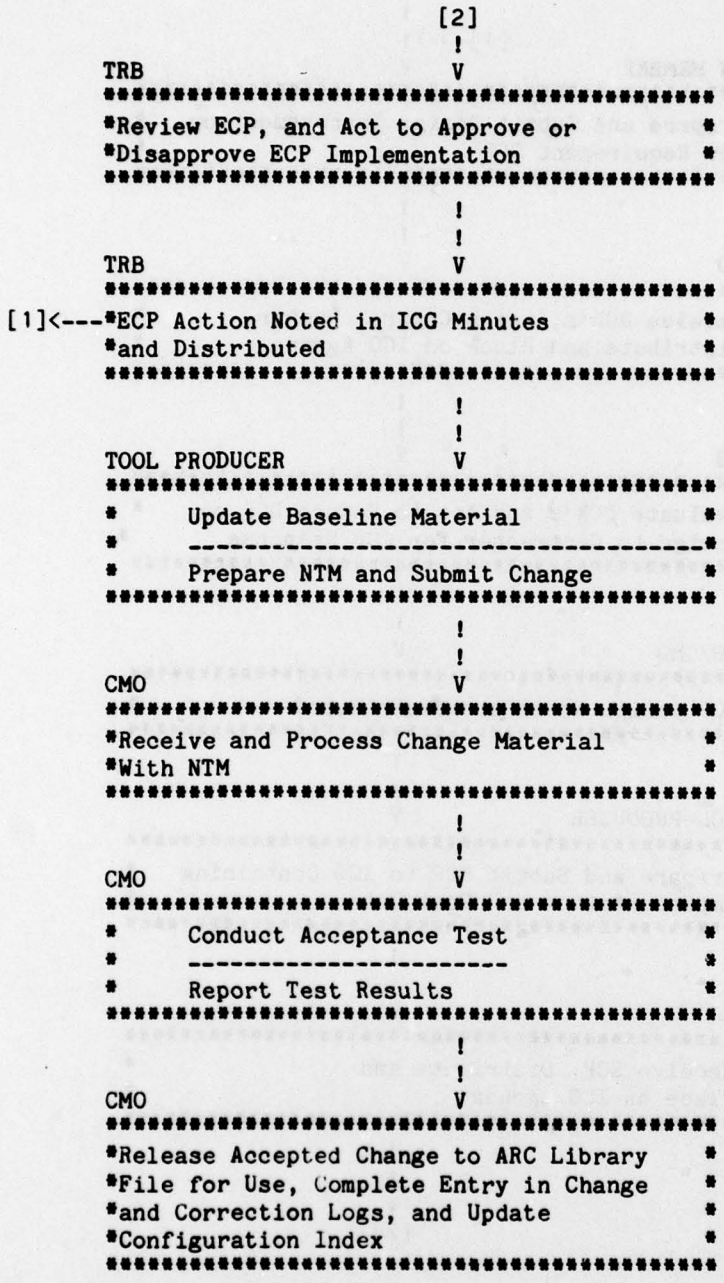


Figure 5-3: NEGOTIATED CHANGE PROCESSING (Continued)

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.