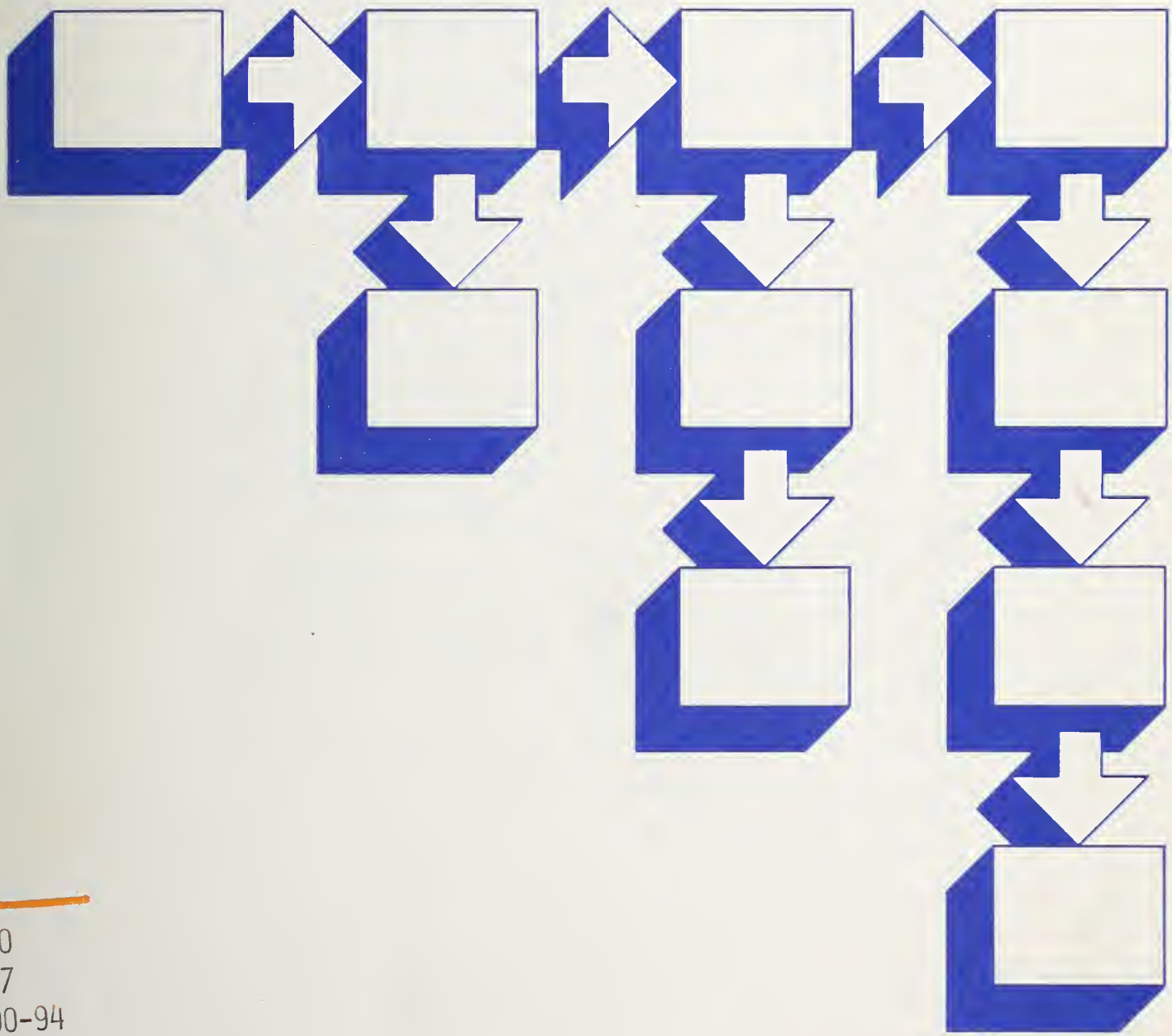
National Bureau
of Standards

Computer Science and Technology

NBS Special Publication 500-94

NBS FIPS Software Documentation

NBS
PUBLICATIONS



NATIONAL BUREAU OF STANDARDS

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

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

Absolute Physical Quantities[2] — Radiation Research — Chemical Physics — Analytical Chemistry — Materials Science

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

Applied Mathematics — Electronics and Electrical Engineering[2] — Manufacturing Engineering — Building Technology — Fire Research — Chemical Engineering[2]

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

Programming Science and Technology — Computer Systems Engineering.

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

Computer Science and Technology

NBS Special Publication 500-94

NBS FIPS Software Documentation

Proceedings of a Workshop
Held March 3, 1982 at NBS, Gaithersburg, MD.

A.J. Neumann, Editor

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

U.S. DEPARTMENT OF COMMERCE
Malcolm Baldrige, Secretary

National Bureau of Standards
Ernest Ambler, Director

Issued October 1982

Reports on Computer Science and Technology

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

Library of Congress Catalog Card Number: 82-600600
Natl. Bur. Stand. (U.S.) Spec. Pub. 500-94, 294 pages (Oct. 1982)
CODEN: XNBSAV

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1982

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402
Price $8.50
(Add 25 percent for other than U.S. mailing)

Acknowledgement

The support and assistance which have made this workshop possible are gratefully acknowledged. The steering committee and the program committee, whose names are listed separately, have worked for over a year diligently and intensely to shape the program and to organize the sessions. I wish to express my appreciation to all committee members, authors, discussants, recorders, session chairs and the organizations, both in the private sector and in Government, who supported the participants. Special thanks are due to Leo Beltracchi, who contributed materially to the planning, organization, and success of this workshop.

A.J.N.

Disclaimer

The viewpoints expressed by participants in this workshop are those of the individuals and not those of organizations, agencies, or individual companies.

Workshop Organization

Workshop Chair: Albrecht J. Neumann, National Bureau of Standards

Program Chair: Leo Beltracchi, U. S. Nuclear Regulatory Commission

Publicity Chair: Virginia C. Walker, U. S. Department of Energy

Treasurer: Bonnie S. Eilertson, National Bureau of Standards

Arrangements: Greta D. Pignone, National Bureau of Standards

Exhibit: Thomas Q. Stevenson, U. S. Department of Agriculture

Steering Committee:
Leo Beltracchi, U. S. Nuclear Regulatory Commission
Charles H. Dickson, Jr., U. S. Department of Agriculture
Trudy Grieb, Hadron, Inc.
Thomas M. Kurihara, U. S. Department of Transportation
Albrecht J. Neumann, (Chair), National Bureau of Standards
Samuel T. Redwine, Jr., The MITRE Corporation
Charles S. Shimkus, U. S. General Accounting Office
Thomas Q. Stevenson, U. S. Department of Agriculture
Virginia C. Walker, U. S. Department of Energy
C. Robert Mangum, U. S. Department of Defense
Saul Zeveler, U. S. Air Force

Program Committee:
Leo Beltracchi, (Chair), U. S. Nuclear Regulatory Commission
Charles H. Dickson, Jr., U. S. Department of Agriculture
Charles L. Gerhardt, U. S. Department of Agriculture
Trudy Grieb, Hadron, Inc.
Raymond C. Houghton, National Bureau of Standards
Lenore S. Maruyama, Library of Congress
Albrecht J. Neumann, National Bureau of Standards
Joseph Psotka, National Institute of Education
Alfred R. Sorkowitz, U. S. Department of Housing and Urban Development
Virginia C. Walker, U. S. Department of Energy

NBS FIPS Software Documentation Workshop, March 3, 1982

INTRODUCTION

This workshop is the result of over a year's work by a group of people who feel that software documentation has an important place in software and overall systems development. The purpose of the workshop was to review current documentation standards and guidelines, to provide a forum for information exchange, and to propose and discuss future guidelines and standards for software documentation.

In order to determine the scope of the workshop, to obtain some idea of what topics would be of interest, and who the audience might be, a "Call for Indications of Interest", was issued, with a deadline of May 15, 1980. The response was generous, and by October 1980, a preliminary program was finalized.

The program was planned to be a regional affair, covering commuting distance from Washington, D.C., Maryland, and Virginia.

In the following pages there are presented introductions by session moderators, papers presented, comments by discussants, recorders, and members of the audience.

It is hoped that the record of this workshop will contribute to the identification and clarification of software documentation issues, will help in solving some of the issues, and will thus contribute to the development of better and more useful information systems.

The final program, as it was presented, has been reproduced, and references to page numbers in the Proceedings provide an index to the presented material.

NBS FIPS Software Documentation Standards Workshop, Wednesday, March 3, 1982

PROGRAM

8:30 a.m. REGISTRATION

9:15 a.m. GENERAL SESSION - OPENING REMARKS

10:00 a.m. BREAK

10:15 a.m. - 12:30 p.m. MORNING PARALLEL SESSIONS

SESSION A: Applying Documentation Standards - Case Studies

The papers highlight various problems encountered by software groups both inside and outside the Federal government as they have attempted to cope with the array of presently existing software documentation standards.

Discussants: Bruce E. Baxter, Internal Revenue Service
Stephen B. Leibowitz, Library of Congress

SESSION B: Documenting for Operation and Maintenance

FIPS documentation guidelines exist for the Initiation and Development phases but not for the Operation phase of the software life cycle. It is during the latter phase that software is maintained, evaluated, and changed as additional requirements are identified. Is there a need for documentation guidance for this phase? This session explores the question by presenting case studies of governmental and non-governmental software, including commercial products. Software certification, change control, configuration management, and end user documentation needs are discussed.

Welcoming Remarks

James H. Burrows

Director Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, D.C. 20234

James H. Burrows, Director of the Institute for Computer Sciences and Technology, welcomed participants to the Software Documentation Workshop. His remarks emphasized the need for software documentation by many users throughout an organization and throughout the software development and maintenance process.

Keywords: documentation needs; FIPS; software management.

On behalf of the Institute for Computer Sciences and Technology (ICST), I am pleased to welcome all participants to the Software Documentation Workshop. ICST has responsibility under the Brooks Act to develop standards and guidelines for the more effective management and use of automatic data processing (ADP) by the Federal Government. One particularly critical area of focus is that of software development and maintenance. To assist agencies in this area, ICST has issued Federal Information Processing Standard (FIPS) publications, reports, and guides for software documentation. Several of these documents are quite recent, and some have been available for some time. We are continuing to identify the user's needs for additional guidance and help in the software documentation process, and we plan to issue additional products in the future.

The fact that this workshop is so well attended suggests that many of you agree that software documentation is needed for more effective use of software resources and that software documentation processes must be improved. I believe that software documentation is vital to the software development and maintenance process, that it is needed throughout the software planning and development cycle, and that it is needed by many people throughout an organization.

No longer is the development of software a one person process. Today most programs are large ones, especially administrative applications, and files are often shared by many people.

Software documentation helps to bring discipline to an ADP activity. It is useful in planning and managing resources, in implementing audits and evaluations, and in planning and implementing computer security procedures. Software documentation also provides continuity to an ADP activity as personnel and needs change. We have all had experience with a program that disappears or stops working when the programmer who developed it leaves. This is costly and inefficient. Another reason why software documentation is needed is to promote common understanding and expectations about the software both within the organization, and, if the software is purchased, between the buyer and the seller. Documentation helps to define what is expected and to verify what is delivered. Further, software documentation provides flexibility within an organization by enabling people to move from one job to another, and to make efficient use of their training.

The individual entrepreneural developers of software certainly need documentation because property rights in copyright and patent right cases are difficult to prove without it. And in the context of the strength of the U.S. software industry, I believe that good documentation is essential to the ability of U.S. companies to compete in international markets.

Software documentation is needed throughout the software life-cycle, starting with the software planning process, and continuing through many stages to serve different users and changing needs. Software documentation is the management trail that lets the software developer and the manager know where they are in the development of a system.

As for who needs software documentation, I believe that the entire organization does -- planners, developers, managers, testers, trainers, maintainers, operators, and users. The number of people involved varies with the size of the systems. Single user microcomputers may have only one person who needs documentation, and that is often part of the system itself. However, for very large systems such as military and airline reservations, formal documentation is essential, and the potential users are numerous.

It is clear that documentation is a necessary, but not sufficient, factor in a well-run and disciplined activity. No engineering discipline exists without documentation; it provides a time-ordered check list to prompt attention to the right things at the right times. Documentation is not a single process for a single user, but a broad-based process that serves different needs and different users. ICST is especially interested in helping Federal agencies identify their needs for documentation and find ways to achieve it.

I wish you a successful workshop and hope that you make progress in addressing this most important issue. We look forward to helping the Federal Government use the ideas and the information that develop in this workshop.

Charge to Workshop

by

Albrecht J. Neumann

Institute for Computer Sciences and Technology
National Bureau of Standards

We are meeting today under the auspices of the Institute for Computer Sciences and Technology to discuss software documentation guidelines. To put our efforts in context, I would like to say a few words about the Institute, the Workshop structure, about software documentation, and guidelines.

The Institute

This workshop is sponsored by the NBS Institute for Computer Sciences and Technology. This organization was created as a consequence of Public Law 89-306, dated October 30, 1965, which is also known as the "Brooks Act". Under this law NBS is responsible for:

-- providing scientific and technological consulting and advisory services to Federal agencies,

-- developing uniform Federal automatic data processing standards; and,

-- undertaking necessary research in the sciences and technologies of automatic data processing computer and related systems.

During the past 15 years almost 100 Federal Information Processing Standards publications (FIPS PUBs), including standards and guidelines, have been published by ICST. Some of the guidelines we will discuss today have been published in this series. As a matter of policy these guidelines are being reviewed at certain intervals. One guideline which is scheduled for review in the near future is FIPS 38 (Guidelines for Documentation of Computer Programs and Automated Data Systems).

Several papers will deal with FIPS 38. Any proposals for new guidelines will be candidates for publication as a FIPs.

Workshop Design

Now I would like to say a very few words about the structure and design of today's meeting. In planning for today's activities we had several options. We could have planned to have a series of lectures, or a series of document reviews by small groups. We instead planned for parallel sessions to provide maximum exposure of ideas, to a large number of people, hopefully eliciting a large number of comments. Since most of the papers will appear in the proceedings you will have an opportunity to read the material at your leisure and at your own pace. We have asked the authors to discuss the highlights of their papers, and have asked the moderators to plan for maximum audience interaction.

We today are discussing Software Documentation Guidelines.

A few comments are in order to define a little more precisely just what is meant by this term. Software covers a broad concept: it includes not only computer programs, but also descriptions and specifications of programs, i.e., documentation or programs, as well as documentation of system functions and instructions for people which are needed to run computer programs. Also included are data used for program and system operation.

Similarly Documentation covers a rather broad area: While computer programs provide instructions to machines on how to solve specific problems, documentation provides a communications medium for people. It is prepared to provide information on why computer programs were prepared, how they are to be designed, what the systems are expected to do, how to interpret processing results, how systems are to be operated, and how they are constructed so that they may be used again, modified, extended, and used by others. Documentation poses a host of problems.

What are some of the documentation problems?

The producer of documentation is faced with a host of technical problems. Documentation serves as a communications medium between a variety of people during the development and operation of a software system. Various document types, addressing different audiences, must be consistent with one another and integrated into an overall management methodology. Documents must be correct at each instant of time and at the same time must incorporate changes, must be easy to update, and must be available when needed. Documents must be easy to use. With all these constraints, documents must be producible on a timely, cost-effective bases.

Analyses of system problems have surfaced shortcomings directly attributable to documentation, or the lack of it:

Millions of dollars have been wasted because of poor user requirements specifications,

Systems were not usable because of lack of system documentation,

Systems were found to be not maintainable because of lack of system documentation,

Funding for needed documentation was not provided,

Also, some of the problems related to documentation can be traced to lack of management interest, support, and control related to software documentation.

and so on . . .

Why do we need guidelines?

It has been shown that intelligent use of guidelines can provide great economic and social benefits, and can solve major problems in systems development, operation, maintenance, and re-use.

Similarly it has been shown that lack of guidance has led to waste of millions of dollars, time, and resources.

Why are FIPS software documentation guidelines useful? They provide for generally understandable communications media which support project management, product development, operations and maintenance, transfer and re-usbility of computer programs. Thus guidelines can save both money and time in all these efforts.

Why do we have this workshop?

I now would like to review the rationale for this workshop:

Existing Information

There are many guidelines and standards in existence which have a bearing on software documentation. They were developed by professional organizations, the American National Standards Institute, government organizations, and the International Standards Organization, to name a few.

Many software users are not familiar with these guidelines and standards, but users need to know about their existence. We will discuss some of these existing standards and guidelines today.

Knowledge about guidelines alone, however, is not enough. Users need to know how to use guidelines, and use them intelligently. In some areas there is a lack of guidelines and new ones may need to be developed. In other areas there is an overabundance of guidelines which often conflict with one another; here direction needs to be provided to decide which guideline should be used.

We will discuss experiences of users with some of the guidelines, discuss pitfalls in application, misuse as well as useful experiences.

New Guidance

There will be need for new guidelines. Our technology is expanding rapidly and new guidelines will be needed to save money, time, and human resources, and to provide proper documentation when needed. The need is to provide high quality software systems, based on quality documentation. The need is to eliminate waste--and to increase productivity.

Research in support of standards development

And finally there is need to do preliminary analyses and to do some real research, to determine needs, impact of proposed standards, balance costs and benefits, and determine development priorities. In addition, many technical problems still need solution.

Therefore:

As we discuss software documentation guidelines -- both existing and proposed guidelines -- we should -- individually and collectively, authors, moderators, discussants, records, and all other participants -- critically review the presented papers and the ensuing discussions and ask the following questions:

In connection with existing standards and guidelines:

> Is the guideline or standard discussed useful; can it be improved? How can it be improved? Can we cite economic or other benefits of its use? What guidance can be given to users to make it more applicable, useful, and effective?

In connection with proposed guidelines:

> Does the topic under discussion have elements which could help to achieve economic or social benefits? What would these benefits be? Should a guideline be developed? How could it best be developed?

In connection with applicable research:

What analyses should be performed in support of guideline development?
Who should perform these analyses: What results are expected?

Charge to this workshop:

To summarize the reason for the workshop -- and the charge to this group:

Considering existing documentation guidelines, let us make specific recommendations on their applicability, use, benefits, and possible improvements, including advice and suggestions as to their usability. Let us make recommendations for specific changes, additions, deletions.

Considering new or proposed guidelines, let us make specific recommendations on their applicability, use, benefits, and possible improvements, including advice and suggestions as to their usability. Let us make recommendations for specific changes, additions, deletions.

Considering new or proposed guidelines, let us identify new areas where they might be useful, and make specific recommendations on what, how and when to document.

Considering analysis and research let us identify directions for research which might lead to new approaches to software documentation and make specific recommendations.

It is hoped that this workshop will be the beginning of a series of steps which will lead to major improvements in software documentation. This, in the broader context, should eventually lead to improvements in software engineering which will reduce system costs and improve system quality and software productivity in the Federal Government.

Measured on a larger scale, our efforts should support economic progress of our industry, improve productivity, and support our national growth as well as our international commitments.

SESSION A: Applying Documentation Standards - Case Studies

Moderator: Charles H. Dickson, U.S. Department of Agriculture

The papers highlight various problems encountered by software groups both inside and outside the Federal government as they have attempted to cope with the array of presently existing software documentation standards.

User Experience and Compatibility in Documentation Standards
A Summary

Betty F. Maskewitz

Engineering Physics Information Centers (EPIC)
Oak Ridge National Laboratory (ORNL)

This paper reviews existing guidelines for documentation of scientific computer programs or data libraries and outlines the essential elements for facilitating exchange of the software. Selected case studies will be made in which accepted standards were followed from the programming stage through documentation, and an analysis of user experience.

Keywords: Documentation standards; Software compatibility; User experience.

1. INTRODUCTION

The problem of computer software interchange is one that has long plagued computer users. Converting a computer program from one computing environment to another is frustrating, tedious, time consuming and costly. The problem is compounded when documentation is inadequate or unavailable. Early informal efforts by individuals who recognized the importance of software documentation to facilitate exchange resulted in the formation of several standards groups under varied sponsorship. One such group, developed under the auspices of the American Nuclear Society's (ANS) Mathematics and Computation Division (M&CD), is today a subcommittee (ANS-10) of the Society's Standards Committee. This standards effort, aimed at scientific computing applications, is the one with which I have been most closely associated. Several standards to facilitate interchange were developed by ANS-10 over the last 20 years.

In the early 1960s we were very cautious in our approach to standardization. The name we adopted (STICE) has long lost its identity, but we pronounced it "sticky."

> *"We look upon standardization effort as a two-edge sword. While it has great potential for cutting through costly duplication, it has equal potential for cutting off further development. This dilemma is particularly acute in the area of computing. On the one hand, standardization could reduce the tremendous expenditures of money and manpower required for the creation of computer programs by increasing the utility of each of them. On the other hand, experimentation with many different languages and systems must be permitted and encouraged if we are to close the gap between hardware development and our ability to use it effectively.*
>
> *"Documentation, however, is one area in which recommendations for minimum standards should be made. In the past, documentation has been directed primarily to the person who prepares input for the program. This type of documentation does permit the program to be used effectively at the originating installation but is generally not sufficient if the program is to be used by others. It is feasible, however, for documentation to be of material help to a programmer in another installation who is incorporating the program into a new operational environment. To this end the quantity of documentation must be increased over that normally supplied. It should certainly include not only the report directed toward the user but also source language comments and installation reports."*

The above premise, part of the foreword of the first proposed standard promoted by STICE, appeared in the first draft circulated for comments.[1] It was entitled, "A Code of Good Practices for the Documentation of Digital Computer Programs." Adopted as ANS-STD. 2-1967, it was revised and reissued as American National Standard ANSI N413-1974 [2] and is now again in the revision process. During this same period progress was also made in parallel complementary standardization efforts.

2. USER EXPERIENCE AND COMPATIBILITY

My long involvement in the standardization process has had profound influence on all my professional activities. The ORNL Radiation Shielding Information Center (RSIC) has always treated computing technology in its subject coverage in the same manner as any other valid technology, insisting that it must be well documented, open to critical examination and to modification, and available for general use. The first and most important criteria, it must be documented.

RSIC had two favorable elements in promoting standardization: a specialized user community that needed RSIC's technology resources and a monthly newsletter. We've come a long way. The examples I've chosen to support the premise that standards make a difference are taken from the RSIC experience.

Monte Carlo methods for forecasting the behavior of radiation in diffusing through matter has been used as a research tool since the advent of large digital computers. It is a complex tool, and its use is more an art than an exact science. Human judgment is an essential element. The early practitioners of the art believed it impossible to document. Transmittal to other than the developers followed only after a one-to-one apprenticeship of days and/or weeks.

My ORNL division invested many man-years in programming Monte Carlo techniques in what was meant to be a reactor research tool. Without documentation, the end result remained unused by other than the developers. Using preliminary drafts of a set of guidelines for documentation to facilitate exchange (vintage 1963-1965), the developers were encouraged to document their modular Monte Carlo code system, O5R. To test their efforts, we featured O5R [3] in a seminar-workshop at ORNL in 1965 and, by invitation, at the OECD Nuclear Energy Agency's Computer Program Library at Ispra, Italy, in 1966. The international radiation transport community accepted O5R as the Monte Carlo code to use for at least 10 years. The code package was disseminated to more than 400 requesters, the heaviest action in RSIC in the late 1960s. The computing technology was obsoleted by later development in 1977. The documentation of the Monte Carlo processes is still requested.

A similar experience in high energy physics can be cited with the National Bureau of Standards (NBS) Center for Radiation Research's Monte Carlo code system for electron and photon transport. Documented in 1968 using available guidelines, [4] a seminar-workshop in 1969 with 53 attendees, disseminated 158 times (1968-1982) all over the world, ETRAN today is used in the nature of a standard in its area of application. It has, of course, been modified (code and document) many times in early usage, less so as time passed. It was last shipped in February 1982.

Another example of standardized software documentation is that represented by the discrete ordinates codes of Los Alamos National Laboratory. The documentation of the two-dimensional, multigroup, transport code systems, TRIPLET [5] and TRIDENT [6] so explicitly follow early ANS-10 guidelines as to be a prototype of the standard. The abstract, the theory, the guide to user application, and programming information sections are each directed toward a specific audience. Just as important, standardized programming practices were also followed.

Documentation, although important, is not the only factor in facilitating exchange of computing technology. Programming practices, [7] program design with user consideration [8] in mind, verification, and ease of modification are important elements and each can be improved through standard practices. Standards can be used to promote effective utilization and enhance reliability of computer programs in any application area, not just the scientific.

There exist several sets of guidelines for the documentation of scientific computer programs. Good documentation promotes understanding, reduces duplication of effort, eases conversion to different computer environments and aids modification for extended applications. Good documentation is needed to facilitate effective usage, transfer, conversion, and modification of computer programs. Good documentation is essential for implementation and effective use of programs within installations other than those in which they are developed.

I was recently involved in the validation of energy-economy models used in predictive calculations for the DOE Energy Information Administration's annual report to congress. As usually happens, the first serious problem area was related to inadequate or nonavailability of documentation of the models. I was asked, at some stage of the validation effort, to review the several documentation standards efforts and suggest guidelines for the modelers to follow.

A brief comparative review was made of FIPS PUB-38, "Federal Information Processing Standards (FIPS)" publication of February 1976 entitled "Guidelines for Documentation of Computer Programs and Automated Data Systems" and ANS-10.3/ANSI N413-1974 "Standard on Guidelines for the Documentation of Computer Programs" and its 1982 revision; and ANSI/ANS-10.5-1979, "Guidelines for Considering User Needs in Computer Program Development." The result of the study is appended.

The RSIC experience in promoting software standards points to several conclusions. The first of which suggests that we can and should cooperate in this important task.

3. CONCLUSIONS

Either or all of the aforementioned guidelines, if followed reasonably closely, would produce better documentation. The fault, then, is not in the guidelines but in the failure of code developers to consider documentation as an important function. Code development evolves in phases from the time that an idea to create the software occurs through the time that that software produces the required output and, beyond that, to the time that there is extensive usage in a wide variety of applications with resulting feedback to a software maintenance group. Four phases apply to the software life cycle: initiation, development, operation, and continuing maintenance until it is no longer used. The documentation of the software should evolve through the same phases. If the documentation develops as the code develops, draft copies should be available during the test stage and continue through the operation and maintenance phases, updated as needed. By the time the code will have achieved "public domain" status, the documentation should be current. It follows that any contract let for code development should include as a vital part of the action plan the necessity to document the code development.

Documentation preparation, therefore, should be treated as a continuing effort, evolving from preliminary draft through changes and reviews, through the documentation and software delivery, and to subsequent updates indicated by user feedback.

There are at least four essential parts to the documentation of computer code systems: 1) the program abstract, directed to the potential user; 2) the application information (user's manual), directed to the individual concerned with the execution of the program to obtain results for his application; 3) the problem or function definition, directed to those concerned with the mathematical models and algorithms employed in the program; and 4) the programming information, directed to the programmer concerned with the implementation, maintenance, and modification of the program.

The above, which should outline the content of the documentation, can be one all-inclusive document for a simple code system or up to four separate volumes for a complex system.

A sampling was made of the experience of an information analysis center (RSIC) which treats scientific computing technology in the same manner as any other valid scientific information, i.e., open to critical examination, widely used, frequently changed to reflect the changing state of the art, and documented following published standards. The 20-year history of the center in which the use of documentation and other standards were vigorously promoted records tangible evidence that the use of standards makes a significant difference in utilization and increased usage serves to advance the state of the art.

Selected case studies were presented in which it was obvious that standards were followed in the original documentation of the code development. It has been amply demonstrated that compatibility with modern software technology and methodology can be maintained by frequent documentation and program review, and resultant updates made through the "open code/data package" concept described below.

Validation of any computing technology in a rapidly changing technological environment, is a continuous process and necessarily requires the dynamic interaction of several elements. RSIC leaders have pioneered in related work and have been cited by their peers as having materially advanced the state of the art in the field by developing and following procedures and techniques designed to promote a standardized product with consequent quality assurance. The "open code/data package" has been the concept through which RSIC has promoted standards. "Open" codes (models/computer programs) and data libraries are those which become well-documented, closely scrutinized by the industry at large, widely used and frequently modified, i.e., open to the same critical examination given

to any valid scientific information. The concept is realized when the developer, the center. and the user collaborate to assess and improve the state of the art. Positive feedback and this close inter-relationship results in changes that are reflected in the "open code/data package" as updated (improved) versions. Dissemination of the packages with a training and consultant service promotes wider usage with consequent feedback to the Center. As long as there is an interest in the problem area, the code package continues to grow in use and effectiveness. The concept serves to promote standardized methods with consequent quality assurance.

The process begins and ends with documentation standards.

4. REFERENCES

1. Nuclear Engineering Bulletin, Volume 4-1, September 1966, pp. 1-8 (A Publication of the American Nuclear Society).

2. American National Standard, ANSI N413-1974, "Guidelines for the Documentation of Digital Computer Programs," June 1974, published by the American Nuclear Society.

3. Coveyou, R. R., Sullivan, J. G., Carter, H. P., Irving, D. C., Freestone, R. M., Jr., and Kam, F. B. K., "O5R, A General-Purpose Monte Carlo Neutron Transport Code," Oak Ridge National Laboratory report ORNL-3622, February 1965.

4. Berger, M. J. and Seltzer, S. M., "ETRAN: Electron and Photon Transport Programs," National Bureau of Standards reports 9836 and 9837, June 1968.

5. Reed, Wm. H., Hill, T. R., Brinkley, F. W., and Lathrop, K. D., "TRIPLET: A Two-Dimensional Multigroup, Triangular Mesh, Planar Geometry, Explicit Transport Code," Los Alamos Scientific Laboratory report LA-5428-MS, October 1973.

6. Seed, T. J., Miller, W. F., Jr., and Brinkley, F. W., Jr., "TRIDENT: A Two-Dimensional, Multigroup, Triangular Mesh Discrete Ordinates, Explicit Neutron Transport Code," Los Alamos Scientific Laboratory report LA-6735-M, March 1977.

7. American Nuclear Society Standard, ANS-Std. 3-1971, "Recommended Programming Practices to Facilitate the Interchange of Digital Computer Programs," April 1971, published by the American Nuclear Society.

8. American National Standard, "Guidelines for Considering User Needs in Computer Program Development," ANSI/ANS 10.5-1979. American Nuclear Society, Hinsdale, IL.

This work is jointly sponsored by the U.S. Nuclear Regulatory Commission under Interagency Agreement 40-548-75, the Defense Nuclear Agency, and the U.S. Department of Energy's Office of Energy Technology under contract W-7405-Eng-26 with Union Carbide Corporation.

APPENDIX

GUIDELINES FOR DOCUMENTATION TO FACILITATE INTERCHANGE OF DIGITAL COMPUTER PROGRAMS

Betty F. Maskewitz
Oak Ridge National Laboratory (ORNL)

There exist several sets of guidelines for the documentation of computer programs. Good documentation promotes understanding, reduces duplication of effort, eases conversion to different computer environments and aids modification for extended applications. Good documentation is needed to facilitate effective usage, transfer, conversion, and modification of computer programs. Good documentation is essential for implementation and effective use of programs within installations other than those in which they are developed.

The following definitions are applicable specifically to this memorandum: (1) Algorithm. A well-defined procedure or process for the solution of a problem to a specified degree of accuracy in a finite number of steps. (2) Benchmark Problem. Both a well-defined problem and corresponding solution, endorsed by a professional society or other recognized technical entity that can serve as a validated reference. (3) External Data Files. The data files which exist prior to execution or after completion of a computer run. (4) Computer Installation Environment. The computer hardware devices and software support that are utilized by a computer program and affect its design and operation. (5) Code Package. All computer-readable and printed material necessary for transmitting and implementing a program (or model) in a different computer installation environment than that in which it was designed.

A. DOCUMENTATION GUIDELINES

There are at least four essential parts to the documentation of computer code systems: 1) the program abstract, directed to the potential user; 2) the application information (user's manual), directed to the individual concerned with the execution of the program to obtain results for his application; 3) the problem or function definition, directed to those concerned with the mathematical models and algorithms employed in the program; and 4) the programming information, directed to the programmer concerned with the implementation, maintenance, and modification of the program.

The above, which should outline the content of the documentation, can be one all-inclusive document for a simple code system or up to four separate volumes for a complex system.

1. Computer Program (Model) Abstract

The abstract provides a summary of the capabilities of a computer program (model), or code package, and the requirements for implementation. The abstract should be concise, but convey sufficient information to permit the reader to assess the applicability of the program to his needs and the effort required to make it operational. It should include a brief statement of: (a) program identification: name, descriptive title, and information necessary to uniquely define the current version; (b) description of problem or function; (c) method of solution: mathematical techniques, procedures, and numerical algorithms; (d) auxiliary routines or external data files required for utilization; (e) limitations imposed by the mathematical model or computer facilities; (f) identification of computer(s) on which the program has been successfully executed; (g) information to enable a user to estimate computer execution time for a typical application; (h) programming languages; (i) operation systems: software system and versions utilized; (j) a list of the computer hardware required for utilization; (k) the names and addresses of the individual(s) or group currently responsible for the code model package; (l) references; (m) material available: the contents of the code package and the procedure for obtaining this material.

2. Application Information (User's Manual)

The User's Manual should be sufficiently detailed to permit effective use of the program (model) and yet concise enough to serve as a referral document for preparation of input data and interpretation of results. It should include general description including a synopsis that conveys the nature of the problem solved, defines the processing tasks performed, and describes the methods and procedures employed. It would be useful to schematically display the flow of the calculations.

2.1. Program Considerations. Detailed information should be given on the following: (a) function of each major program option; (b) alternate paths which may be dynamically selected by the program from tests on calculated results; (c) restrictions on the range of values of variables; (d) the dimensions of data arrays; (e) dependence of data storage requirements on problem input parameters; (f) the values assigned to constants built into the program; (g) restart and recovery procedures; (h) programmed diagnostics and their causes; (i) any man-machine interactions; (j) information to estimate execution time; (k) any special forms of output; e.g., microfilm, cathode ray tube display.

2.2. External Data Files. External data files should be described as follows: (a) outline the general contents and organization of each external data file; (b) relate the usage of data files to the execution of the program; (c) reference available auxiliary programs which create, modify, or edit these files; (d) reference sources of fixed or permanent data.

2.3. Input Data. General and specific considerations should be given to input data. Generally, one should describe: (a) special input techniques and requirements; e.g., blank field treatment, order of items, field delineation; (b) the handling of consecutive cases (giving conditions of data retention or reinitialization for the next case); (c) the general conventions governing default values.

Specifically, for each input variable one should give variable name, description or definition, format, dimensional units of variable, and the default value if appropriate.

The operating system control commands (cards or statements) required to execute the program should be given with an indication of interdependence with input options and data files.

2.4. Output. The program output should be described with relationships shown; e.g., edited output to input options and output to appropriate equations. Any normalizations of results should be described and associated dimensional units listed.

The physical problem and associated data files should be described and the input data and results presented.

2.5. Sample Problems. The following should be considered in selecting sample problems: (a) choose a benchmark problem or a well-defined example; (b) exercise a large portion of the available programmed options; (c) use only a reasonable amount of computer time.

The following should be considered in presenting the edited output: (a) provide representative output for the options exercised (detailed output can be transmitted on tape); (b) present results of key items in concise form; (c) indicate precision of results.

Report computer execution time for the sample problems giving central processor time, peripheral processor time and elapsed (clock) time, and channel use applicable and available.

3. Problem of Function Definition

The problem or function definition information should convey a thorough understanding of the theoretical and mathematical foundations, referencing the open literature where appropriate. It should define the problems solved, describe the mathematical model employed, and document the computational algorithms and numerical techniques implemented in the program (model). It should specifically include a comprehensive description of the problem solved or of the data processing functions performed. The description of the physical theory in terms of a mathematical model should be reasonably self-contained.

Sources for the model and the mathematical formulations should be referenced. Sufficient detail is needed to permit a user to judge the suitability of the model for application to a particular situation. Assumptions should be noted and information given about limitations.

3.1. Algorithms and Numerical Techniques. The computational algorithms used to obtain numerical solutions of mathematical equations should be described and references to algorithms and numerical techniques provided. The precision of results obtained by important algorithms and any known dependence on the particular computer facility should be described. Unusual features of techniques used should be described. For iterative solutions, the use and interpretation of convergence tests and recommended values of convergence criteria should be included. For probabilistic solutions, the precision of results having a statistical variance should be discussed.

3.2. Data Sources. Background information about source, contents, and use of data libraries should be provided.

4. Programming Information

The programming information is directed to the individual responsible for implementing the program on his computer, modifying or extending it to meet local needs, or converting it to a different computer environment.

Reference may be made to appropriate items described in other sections of the program document. Give further information as necessary, to explain the programming details. The citing of computer-produced documentation that is generally available may complement traditional documentation. Examples of such computer-produced documentation are: a listing of the source program that contains carefully composed comment cards, a cross-reference dictionary of subroutine names and entry points, and flowcharts of the program logic.

4.1. Source Program. The source program description should include: (a) identification of the source language(s) of the coding; (b) a flowchart showing the overall program structure and logic, and detailed flowcharts where appropriate; and (c) an indication of known areas of dependency upon the local computer installation support facilities.

Detailed information should be given as follows: (a) define the role and function of the main program and each subprogram (identify argument lists and their use); (b) for a particular subprogram, indicate those routines which call it and, in turn, those subprograms it may call; (c) relate the problem variables and constants to the program mneomics; (d) describe shared storage assignments; e.g., COMMON in FORTRAN; and (e) describe in detail the functions performed by machine-dependent subprograms that are unique to this program.

4.2. Data Files. The computer-oriented details of temporary and external data files should be provided: (a) specify the names, usage (input, output, or scratch), structure, mode and data elements of each data file; and (b) discuss program procedures related to the use and maintenance of data libraries and files, giving data file retention and allocation requirements.

4.3. Hardware and Software. Logical devices used should be enumerated and the use of each device and any associated data blocking schemes described. The contents and format of the information resident on each device should be identified.

The computer installation environment in which the model is normally executed should be described as follows: (a) list the machine configurations on which the model has been tested successfully; (b) enumerate the main memory storage requirements, the amount and type of auxiliary storage (drum, disk, data cell, tapes) and the peripheral equipment (punch, printer, plotter); (c) identify any special hardware utilized; e.g., clock, on-line communication channel.

In addition, the software requirements should be given: (a) identify the operating system, language processors, and associated subroutine libraries invoked by the model, citing the manufacturer's appropriate versions and releases; (b) describe known deviations from the manufacturer's supported software that are required by the model; e.g., local mathematical and utility routines, and other installation-dependent software.

4.4. Programming Considerations. Certain programming considerations are important for successful implementation: (a) explain the system-control commands required to execute the program; (b) document the overlay or segmentation scheme if one is used; (c) describe the storage allocation and data-management procedures; indicate the problem-dependent nature of the memory requirements; discuss program alternatives which affect data storage and use of data buffering; e.g., variable dimensioning; (d) discuss the restart, recovery, and successive case capability.

B. CODE PACKAGE TRANSMITTAL FORMAT

The code package is the aggregate of card decks and printed material associated with a computer program (model), including an abstract of the complete code package for publicizing availability. The package should include:

1. Computer-readable material written on magnetic tape as separate files: (a) source (main) program in the form of card image records; (b) any auxiliary code used to generate input for the main program or analyze results; (c) any external data libraries; (d) input data for sample problem(s); (e) system control cards for execution of the program; (f) output from running the sample problems in line-printer format.

2. Printed Materials: (a) master tape list describing each file in (a) and how each is written on tape; (b) documentation of the complete code package; (c) abstract of the code package (in standard format) for publication.

C. VERIFICATION/VALIDATION RECORDS

Initial verification of the code system is normally done by the code developers and the results documented in the original report (see item A). Validation is normally a separate effort, independently done and reported. As validation results become available, they should also be included in the code package, as defined in item B. These records should include: documentation - published results of findings, audit trail on data used, and record of experience gained from running benchmark or other calculations.

NBS FIPS Software Documentation Workshop.
March 3, 1982

Systems Development Methodology and Documentation Practices

Louis J. O'Korn

Manager, Systems Development
Chemical Abstracts Service

The approach Chemical Abstracts Service (CAS) has taken to prepare and manage the full range of software documentation will be described. For each development stage this presentation will summarize the deliverable items of documentation, specific standards and procedures guiding the documentation process, and specific tools supporting the preparation and management of documentation.

1. INTRODUCTION

All methodologies are based on the assumption that you can analyze a process and identify a consistent set of tasks that must be performed in a particular order to complete the process. Similarly, in systems development, the methodology attempts to predict the tasks that must be performed to design, develop, and implement a computer system. All effective systems development methodologies recognize that there are actually two types of activities that must be performed to build and operate a computer-based system:

- Activities to create the proposed system. These include interviewing users about proposed system requirements, documenting objectives and constraints of the proposed system, designing overall system flow for the proposed system, coding and testing programs, and so forth.

- Activities to manage the system building process. These include tasks for planning, organizing, and controlling systems development projects.

The Chemical Abstracts Service (CAS) systems development methodology draws from both these areas. This paper identifies the tasks required to build a system. It describes the phases of the CAS system life cycle, specific deliverables, and guidelines for each subphase. The project management policies, procedures, and deliverables required to plan, organize, and control the project activities necessary for building computer-based systems are not addressed in this paper.

2. SYSTEM LIFE CYCLE

At CAS, system life cycle activity is described as fitting into one of four broad phases: System Initiation, System Design and Development, System Implementation, and System Operation.

The objectives of the System Initiation phase are to identify the purpose and description of the system and the needs for development resources, to fully document user requirements, to generate a general approach for the proposed system, to study alternate design strategies, and to recommend the best approach for the development of a particular system. This may involve breaking the system into individual components or subsystems each of which would follow the system life cycle.

The next step, System Design and Development, resolves all design features of the proposed system. To do so, all input, output, data base and internal processing aspects of the design are developed by the project team, technically reviewed within Research & Development (R&D) and reviewed by operations and user personnel. Once this review is completed, all approved features are ready for development and are not subject to further design modification without formal review and approval at the design changes. The next objective of this phase is to develop the detailed design and to transform the design into functional system components through program design, coding, and testing.

The third phase of the system life cycle is System Implementation, which includes preparing a system test plan, conversion plans, and user manuals, conducting the system test, performing necessary conversions, releasing the system to data processing operations, creating live files, training users, and beginning operations. The final objective of this phase is to secure formal acceptance of the new system from the user group.

Finally, during System Operation, the objective is to maintain an efficient operation of the system, evaluate it, and make changes as additional requirements are identified.

While these four phases are useful in broadly visualizing a project, they are too general to serve as a basis for planning, scheduling and controlling the work effort. They are further divided into a number of subphase:

Phase 1 - System Initiation

- Project Initiation
- User Requirements Definition
- System Definition

Phase 2 - System Design And Development

- Preliminary Design
- Detail Design

Program Development

Phase 3 - System Implementation

Implementation Planning

System Testing

Operations Start-Up

System Acceptance

Phase 4 - System Operation

3. SYSTEM INITIATION

3.1 Project Initiation

Activity in this subphase is directed toward translating a requested project into a brief Project Initiation Description. Prepared by the Development Project Manager and system requestor, this description gives the name of the project and the user departments concerned and details the project's purpose, scope, benefits, dependencies, background, preliminary development forecasts, and schedules.

If the Project Initiation Description is approved by R&D and the requesting division management, a project then enters the User Requirements Definition subphase. When a proposed system is intended to supplement or replace an existing system -- either manual or automated -- the project team begins by reviewing the operation of the current system and assessing its merits and shortcomings. Activity then focuses on determining the users' requirements for the new system, working closely with the user staff. The results are rigorously documented to guide subsequent design and development tasks.

3.2 User Requirements Definition

The User Requirements Document represents the key input to the subsequent design stage because it is a definitive statement of the functions to be performed and outputs to be delivered by the system as seen by the user. Essentially, it is a playback: the result of considerable dialogue between the analyst and the user regarding the functional specifications of the system. It is important to note that the user can be the eventual system user, a user-appointed representative, a management-appointed task force, or a new venture team; the specific approach is determined on a project-by-project basis.

The User Requirements Document includes an elaboration of data element requirements, identification and description of required algorithms, clarification of policy with its known impact and implications, and definitions of important concepts. It includes functional flows only as needed to illustrate the above material and clarify interfaces with existing systems. Generally, this document will be co-authored by the analyst and the

user, or a user representative, and will be subject to review and approval by the user and R&D management.

3.3 System Definition

Once the user requirements are approved, the system enters the System Definition subphase, where the user requirements are used to prepare a general design of the new system. The project team identifies and evaluates as many design alternatives as practical while ensuring that enough alternatives have been considered to respond to critical needs. Criteria for evaluating the alternatives might include such factors as anticipated system benefits, number of features supported, development and operation costs, or technical feasibility. Before proceeding to the next phase, the project team examines the design alternatives and the criteria used to make a final evaluation and recommends an approach for developing the system. The proposed system is set forth in a System Description which is a user-oriented document describing a proposed approach, alternative approches, and evaluation criteria. In addition, a Cost/Benefit Analysis may be incorporated into the System Description or provided as a separate document. The proposed approaches must be approved by R&D and user management.

4. SYSTEM DESIGN AND DEVELOPMENT

4.1 Preliminary Design

Completion of the three System Definition subphases marks the beginning of the System Design and Development phase. Its first subphase, Preliminary Design, focuses on designing and documenting a preliminary version of the proposed system. The tasks performed during this subphase include preparation of an overall system flow, design of data base content, identification of the various subsystems, creation of narrative descriptions of systems operation, and description of system inputs and outputs. Most importantly, the preliminary design reflects the users' processing requirements for the new system. The user staff is involved in review of the preliminary design to interact with the design process and ensure all requirements are included.

The project leader prepares the Preliminary Design Document, which includes a summary of features, system architecture, flow and narrative function descriptions, data definition summary, transaction summary, report summary, functional responsibilities, interfaces, performance estimates, etc. This document is the basis for a technical design review within R&D to ensure soundness of the technical approach, and an in-depth functionality with all users to assure that all requirements are satisfactorily met. Included as separate documents, or as sections in the Preliminary Design documentation, are updates to the User Requirements and Cost/Benefit Analysis. The Preliminary Design requires the approval of R&D and user management, and provides guidelines for subsequent technical development in the Detail Design subphase.

4.2 Detail Design

Activity in this subphase is devoted to designing detailed versions of every systems component identified in the preceding subphase. Major tasks include preparing and documenting the detailed report layouts, designing all input forms and screens, identifying the final data base structure and content, designing controls for all manual and computerized aspects of the operation, file backup and recovery procedures, data entry procedures, online dialogs, etc. Included as separate documents or as separate sections in the Detailed Design documentation, are updates to the User Requirements and Cost/Benefit Analysis.

The entire system features reflected in Detailed Design are now intensively reviewed by the system users. This user review marks a major milestone in the project, because all design elements approved here will become permanent features of the system. Once all the problems have been resolved, changes in design during subsequent subphases will be formally documented and require final approval by R&D and the user management. If during the Preliminary or Detailed Design subphase a serious design flaw is identified, it may be necessary to recycle through various subphases to reconsider aspects of the approach taken.

4.3 Program Development

It is essential that application programs be designed to take maximum advantage of sophisticated hardware, software, systems and capabilities. In this subphase, the structure of schedules, programs, and modules is developed, documented, and thoroughly reviewed by qualified technical personnel. Program design produces program structure which is reviewed through walk-throughs. Program structure is converted into functional components through program design, coding, testing, and documentation. Programming is performed in accordance with corporate standards and policies. During testing, emphasis is placed on rigorous preparation of a test plan describing all anticipated program responses to test situations, and when testing commences, test results will be compared to these test plan predictions, thus providing an objective measurement of requirement delivery. All test results will be thoroughly reviewed to ensure that system components are operating satisfactorily. This subphase produces a completed operational program, program documentation, and operations documentation.

5. SYSTEM IMPLEMENTATION

5.1 Implementation Planning

The System Design and Development phase and System Implementation phase overlap at this point in the development cycle. Implementation Planning begins immediately after completion of the Detailed Design subphase and runs parallel to the Program Development subphase. The subphases overlap in this way so implementation can begin immediately upon completion of program testing. Included in this subphase are tasks to develop detailed conversion plans, system testing plan, training schedules, and supporting materials. Tasks are also performed to develop many of the user manual materials for subsequent

implementation training and system operation. During their review of these products, the users are required to commit themselves to training and conversion schedules as set forth in the implementation plans.

5.2 System Testing

During the System Testing subphase, tasks are performed to assemble programs into subsystems and to integrate these subsystems into a working system. Work begins with the creation of extensive test scenarios and test data is then prepared to simulate these situations. Operations Documentation is compiled, reviewed and finalized. The systems test is conducted in accordance with the test plan. As with the program test, system test results are compared with the test plan predictions to ensure that the system is operating properly. As part of the systems test, the users conduct an acceptance test to verify that their specifications have been met.

5.3 Operations Start-Up

This subphase marks the transition of the system from "in development" to "operational" status. Here, tasks are performed to finalize operations documentation, data entry documentation, user manuals, schedules, etc. All documentation is reviewed with appropriate user/operations groups to facilitate the turnover and start-up processes. Various user and operational areas certify that the documentation meets company standards and that the system is ready to begin routine operation.

Tasks are then performed to finalize implementation schedules. The users are given manuals describing external procedures, and are thoroughly trained to use their new system. Finally, files are set up or converted to support ongoing system operations.

5.4 System Acceptance

After a system becomes operational, the project team is responsible for monitoring operations and assisting the users wherever problems are encountered. If a new system was developed to replace an older system, the two may be run in parallel until all parties are satisfied that the new system is functioning correctly. At the end of the parallel operations and before the old system is retired, users are required to formally certify that their training, documentation, and the new system itself satisfy the original requirements. At this stage, all project documentation is transferred into permanent storage and an evaluation is made of the project on the basis of observed strengths and weaknesses.

6. SYSTEM OPERATIONS

With the acceptance and routine operation of the system, staff maintain installed software and make minor enhancements as necessary. Documents produced during this phase describe problem resolution procedures e.g. abnormal program completions, corrections

actions, condition-causing problems, requests and documentations for minor system enhancements, and documentation of program module changes.

7. SUMMARY

There is a wide variety of documentation produced during the CAS system life cycle. The earlier section summerizes activities performed and key deliverables for each subphase.

To aid the staff in preparation of these documentation items, each item has suggested contents, specific staff responsible for preparation, and specific staff responsible for review and approval. It must be recognized that there are suggested documentation items to identify critical documentation requirements and achieve a consistent CAS scheme. For a particular system, the specific documentation items to be prepared will be proposed by the project leader, and the documentation package may require additional documentation items, the combination of several of these items, an expansion of an item, change in format, etc.

CAS has a separate operational unit responsible for the administration of computer system and data base documentation in support of CAS development, maintenance, and operational activities. The data base function controls the authoritative record and inventory of CAS files, their definition and interrelationships. As part of this data base responsibility, this unit's services include registration of data elements and data sets, recovery and reorganization of system files, management of the file archiving process, management of tape and disk resources, and analysis and consultation on data base performance.

The computer system documentation function manages the master record of computer system and program documentation. It stores the complete inventory of documentation, maintains a historical change record, provides reference library services, processes all program changes, and controls updates to production software libraries.

Experience in Application of Software Documentation Standards

William Bryan
Stan Siegel

CTEC, Inc.

This paper summarizes the authors' recent experience applying software documentation standards contained in MIL-STD-483, MIL-STD-1679, and DoD STD 7935.1-S (and its non-DoD counterpart FIPS PUB 38). Several software documentation problem areas are discussed. One problem area is redundancy -- the requirement to put the same material (in different form or degree of detail) in two or more documents in the same set. Not only does this redundancy increase project costs and lengthen schedules, it also greatly complicates the life cycle maintenance of the documents. A second problem area is the telescoping of test documents in FIPS PUB 38 and DoD STD 7935.1-S. In these two standards, test procedures and test plans are included in the same document. If test procedures are written concurrently with the test plan, customer modifications to the test plan may partially invalidate the test procedures. A final problem area is that of tailoring software documentation requirements. Software documentation standards should permit sufficient tailoring to cope with project size and complexity without vitiating the documents. The paper includes recommendations for improvements to software documentation standards.

Keywords: Product assurance; Software maintenance; Testing; Traceability; Visibility

1. INTRODUCTION

The past few years, the authors have performed software product assurance services for both in-house and external software development projects. Product assurance entails, in our view, the integrated performance of the following four disciplines:

Quality assurance -	the assessment of whether a software product conforms to a pre-specified standard
Verification & validation -	the assessment of whether a software product logically follows from its predecessor software products and is congruent with the software requirements
Test & evaluation -	the assessment of whether a software product satisfies its requirements through testing in a live or nearly live environment
Configuration management -	the set of procedures for visibly, traceably, and formally controlling software products throughout the software life cycle

Performing software product assurance services clearly involves the use of software documentation standards. This use is not connected with the actual development of software and its associated documents, but rather is concerned with the utilization of standards as a tool to assess the successful development of software products.

Our work in software product assurance has exposed us to various software documentation standards and given us experience in their use as product assurance tools. From this experience, we have observed a number of problems in the use of the software documentation standards. In this paper, we address three of these problems -- (1) redundancy among documents, (2) the telescoping of test documents, and (3) the tailoring of software documentation requirements.

In the balance of this paper, we first present a summary of our recommendations regarding software documentation standards. Then we present in turn each of the three problems stated above. We close the paper with our conclusions based on our analysis of these three problems.

2. SUMMARY RECOMMENDATIONS

Based upon our experience with various software documentation standards, we make the following recommendations:

a. Eliminate redundancy among the documents specified by a software documentation standard.

b. Stress in software documentation standards the need for continual maintenance of the documents produced.

c. Modify FIPS PUB 38 and DoD STD 7935.1-S to specify that two pre-test documents, a Test Plan and Test Procedures, be sequentially produced.

d. Modify software documentation standards to require substantial conformance to their format and content specifications, while allowing tailoring and scoping of the various documents to match project size and complexity.

3. REDUNDANCY AMONG DOCUMENTS

In the range of document types specified by FIPS PUB 38 [1] and DoD STD 7935.1-S [2], considerable redundancy is specified. The existence of this redundancy is specifically addressed and explained in both standards. One reason that redundancy is provided is to make the documents "stand-alone" with a minimum of need for cross-referencing (FIPS PUB 38, paragraph 2.3; DoD STD 7935.1-S, paragraph 1.3.4). That is, the basic assumption seems to be made that the reader of a document does not have ready access to other documents produced during a software development project, and possibly may never have read them. In this circumstance, each document must be as complete and detailed as is possible, and must accurately reflect the state of the software at the time of document publication.

This form of redundancy is particularly evident in the Functional Description (FD), System/Subsystem Specification (SS), and Program Specification (PS) sequence found in FIPS PUB 38 and DoD STD 7935.1-S (see Table 1). In the content guidelines of the latter standard, explicit reference is made to paragraphs in preceding documents where related material is found, coupled with statements that "changes...from the next higher order [i.e., immediately preceding] document will be explicitly identified" (see, for example, the content guidelines for paragraphs 2.2.1 and 2.2.2 in Figure 3-04 on page 3-33 of DoD STD 7935.1-S). Such statements imply that a given document (say, an FD) may or may not be accurate once a successor document (say, an SS) has been published. Where and to what extent a predecessor document may be inaccurate is not evident to a reader. A further implication here is that a given document is not maintained once a successor document is published.

It should be noted that 100% redundancy does not exist among these documents. Each document has certain unique sections and paragraphs not found in its immediate predecessor. Thus, from a project viewpoint, the documents are not truly "stand-alone." A reader wishing to become familiar with a project should read the latest document in this sequence, but must realize that he is not accessing all the

Table 1. Redundant Paragraphs Among Several Documents Specified by DoD STD 7935.1-S

Functional Description	System/Subsystem Specification	Program Specification
--	2	2
--	2.1	2.1
3.1.1	2.2.1	2.2.1
3.1.2	2.2.2	2.2.2
3.2	2.2	2.2
--	2.3	2.3
4	3	3
4.1	3.1	--
4.2	3.2	3.1
4.3	3.3	3.2
4.4	3.4	3.4
--	3.5	3.5
--	4	4
--	4.3.1	4.2
--	4.3.2	4.3
--	4.3.3	4.4

pertinent information on the project. Whether this process is satisfactory to him depends upon his purpose in reading about the project.

This redundancy in FIPS PUB 38 and DoD STD 7935.1-S has had a long history. As indicated in Kurihara et al [3], FIPS PUB 38 was derived from DoD Manual 4120.17M [4]. The foreword of DoD STD 7935.1-S states that it supersedes the same manual. The basic format and content guidelines of all three of these standards are nearly identical. However, the DoD manual was not an original document, being based on, and almost identical with, a Navy Department instruction which, in turn, was almost a direct copy of an ADP documentation standard issued by the Naval Command Systems Support Activity (NAVCOSSACT -- now named NARDAC) in 1966. [5] This original standard was designed to be used in the environment in which NAVCOSSACT developed software. In this environment, military tours of duty for users and buyers approximated the development cycle of software (i.e., about three years). In many cases, the originator of a request for a software project would be rotated in his military duties and would no longer be with a command when the end-product software was delivered. Managers at all levels monitoring software development projects faced similar rotations. In such an environment, stand-alone documents were a necessity.

Today the concepts of this original standard provide guidelines for a much broader community, one which encompasses the entire federal government. For the majority of the members of this community, the correlation between changing buyers and users and the software development cycle does not necessarily exist. Thus, stand-alone documents generally are not required. For this broader community, the redundancy specified in FIPS PUB 38 and DoD STD 7935.1-S unnecessarily causes problems through the life cycle of a software project. This life cycle includes the period of maintenance of the software as well as its development. Typically, the maintenance of software includes the correction of latent errors discovered in the operational software and the enhancement of the software to produce new or modified capabilities. This maintenance activity is hampered by the redundancy in these two standards, for, as we subsequently explain, the redundancy limits the visibility and traceability of the software, which are necessary ingredients for effective software maintenance.

This limitation in visibility and traceability is evident from consideration of two observations made in preceding paragraphs: the documents produced under these standards are not truly stand-alone, and once produced, are not maintained. The only reliable document in a series is the most recent one. If previous documents are not consulted, visibility of the non- redundant part of those predecessor documents is lost, which may

have considerable impact on maintenance activities. On the other hand, consultation of the predecessor documents may have an unfavorable effect on software maintenance, since the documents are of questionable reliability. Traceability through the sequence of documents is impaired or destroyed.

The aforementioned objections could be overcome by maintaining all of the documents in the sequence, so that each one is always up-to-date and thus reliable. But such an endeavor greatly increases documentation costs.

A strawman document termed a Comprehensive Document (CD) has been suggested as a general approach to revision of FIPS PUB 38. [3] The CD would be a data base (maintained current) of all information relevant to the definition, design, coding, testing, operation, maintenance, and modification of the system being documented. In this concept, the principal software documents (e.g., an FD or PS) would be regarded as extracts from the CD data base. The CD approach has many noteworthy benefits, but also has a potential weakness in its application: the publication of the principal documents as extracts from the CD data base might result in a situation identical to that created by the FIPS PUB 38 redundancy. For example, the FD, SS, and PS might all contain the current CD paragraph on, say, system functions. Since they are published over a period of time, and since change is endemic to software projects, each document might contain a different definition of system functions. On the other hand, if the documents extracted from the CD data base were not redundant (for example, system functions might only be extracted for the FD), then the problem arises as to how to promulgate changes to elements of the CD data base. One solution to this particular problem would be to maintain and republish each extracted document when changes occur; another would be to publish the CD and changes to the CD rather than publish extracted documents. Further consideration of this strawman document should definitely consider this potential problem in order to avoid continuation of the difficulties caused by the deliberate redundancy specified in FIPS PUB 38.

4. TELESCOPING OF TEST DOCUMENTS

Both FIPS PUB 38 and DoD STD 7935.1-S provide guidelines for two test documents -- a Test Plan written during the design and programming stages, and a Test Analysis Report written at the end of the test stage. The purpose of the Test Plan is "to provide a plan for the testing of software; detailed specifications, descriptions, and procedures for all tests; and test data reduction and evaluation criteria" (FIPS PUB 38, paragraph 1.4.9). The purpose of the Test Analysis Report is "to document the test analysis results and findings, present the demonstrated capabilities and deficiencies for review, and provide a basis for preparing a statement of software readiness for implementation" (FIPS PUB 38, paragraph 1.4.10). The Test Plan is the sole test document preceding the testing of the software; the Test Analysis Report is the sole test document following the testing.

In the environment at NAVCOSSACT referred to earlier, where the progenitor of FIPS PUB 38 and DoD STD 7935.1-S was created, this test documentation schema was useful and satisfactory. NAVCOSSACT developed software and a Test Plan concurrently. When development of the software code was completed, NAVCOSSACT personnel went to the user site, conducted a test, and submitted a Test Analysis Report. The user and the buyer (more properly, in that environment, the sponsor) had no involvement with development nor approval of the testing process.

In the broader community served by FIPS PUB 38 and DoD STD 7935.1-S, users and buyers tend to be more involved in the testing process. The Test Plan generally is submitted to the buyer and user well before testing begins and is approved by the user and/or buyer before any testing is conducted. If the Test Plan does not meet the buyer's approval, it must be revised until the buyer and seller can mutually agree on its contents. In this environment, the monolithic Test Plan called for in FIPS PUB 38 and DoD STD 7935-1.S is not satisfactory.

The problem with the Test Plan is that it is too comprehensive: it contains not only the plans for testing of the software code, but also the detailed procedures for conducting all tests. If, when the Test Plan is submitted for approval, the buyer objects to the planned testing approach, not only must the plan portion of the Test Plan be revised, but part (or possibly all) of the test procedures must also be revised. In essence, it is generally not economically justifiable to expend resources on creating test procedures until test plans have been approved.

In MIL-STD-1679 [6], three pre-test documents are identified and separate Data Item Descriptions are defined for each document: (1) DI-T-2142 for test plans, (2) DI-E-2143 for test specifications, and (3) DI-T-2144 for test procedures. MIL-STD-1679 specifies that all test plans, specifications, and procedures shall be subject to review and approval by the procuring agency. MIL-STD-1679 implies a sequence in producing these three documents. The sequence is made more explicit in the Data Item Descriptions. The development of three separate pre-test documents appears to be more cost effective than the development of a single one as specified in FIPS PUB 38 and DoD STD 7935.1-S. Yet the cost effectiveness is not guaranteed: in a recent project, the authors' company was tasked to produce Test Plan and Test Procedures documents in accordance with MIL-STD-1679, with both documents due the same day} By destroying the sequential nature of these documents, the procuring agency specified a requirement identical with that of DoD STD 7935.1-S. In the project cited, through negotiations with the procuring agency, the delivery dates of these two documents were shifted so as to make them sequential.

When the use of FIPS PUB 38 or DoD STD 7935.1-S is prescribed for a project, the authors' company invariably produces a Test Plan in which the section on test descriptions (Section 4 in FIPS PUB 38 and Section 5 in DoD STD 7935.1-S) states that detailed test procedures will be separately and subsequently produced. These procedures are then developed following approval of the Test Plan by the procuring agency. In this fashion, wasted effort producing unnecessary or undesired test procedures is avoided. This is a practical solution to the problem of telescoped testing documents. It would be preferable to modify these two standards to provide guidelines for a Test Plan and a separate Test Procedures document.

5. TAILORING OF SOFTWARE DOCUMENTATION REQUIREMENTS

The various governmental software documentation standards vary widely in the degree of flexibility afforded to the documentor in using the standards.

At one extreme are MIL-STD-483 [7] and MIL-STD-1679 which are relatively inflexible. MIL-STD-483 states the following in Appendix VI (Computer Program Configuration Item Specification), Section 60.4:

> All paragraphs in this appendix preceded by the designation "Note:" are for guidance only. All other sections of the appendix are requirements for the preparation and control of computer program specifications. Contents of the specification shall be arranged in accordance with the format and paragraph headings described herein. Deviations from the requirements of this appendix require approval of the procuring activity.

The appendix contains four notes, all indicating that additional material or references can be included in a specification. The appendix also contains the following four paragraphs specified as optional for inclusion in a CPCI Specification:

CPCI Specification Paragraph Number	Title
Part I, Section 6	Notes
Part II, Paragraph 3.7.3	Data Base Location Requirements
Part II, Paragraph 3.11	Program Listings Comments
Part II, Section 6	Notes

All of the remaining 71 paragraphs and sections for a CPCI Part I and Part II Specification are required by MIL-STD-483.

MIL-STD-1679, in Section 6, specifies that, if the procuring agency desires to order data that it has had created through work tasking, it must use the pertinent Data Item Descriptions (DIDs) from a list of 17 of them. Each of these DIDs states that the document it pertains to shall be in accordance with the DID's content and format instructions. These DIDs do not contain optional paragraphs, other than some appendices. Only the DID for a Software Quality Assurance Plan (DI-R-2174) implies that non-applicable subjects and items may be omitted. The only flexibility provided under MIL-STD-1679 is whether a document should or should not be included in a Contract Data Requirements List.

An impact of this lack of flexibility might be a reduction in the number of documents produced for a project. In a survey of large aerospace firms primarily developing software for the federal government, Lehman [8] reported that development of over nine different document types was specified on the average for projects of $20 million or more, while development of only six different document types was required on the average for projects of less than $1 million. (Document types in this survey included both source code listings and object code listings and tapes; these two document types were the ones most frequently required for projects.) The requirement that a project use an inflexible documentation standard might be a possible explanation of these results. The cost of creating a document under an inflexible standard is less than linear with respect to project size. That is, the creation of a document under such a standard costs only slightly less for a small project than for a large one. In this circumstance, the number of documents to be produced on a small project might be reduced to keep documentation costs in perspective.

In the other direction, FIPS PUB 38 and DoD STD 7935.1-S provide considerable flexibility to the documentor. Paragraph 3.3.2 of DoD STD 7935.1-S allows any paragraph or subparagraph of a section to be omitted and additional paragraphs to be freely added. In practice, this provision allows a documentor to substitute a completely different partition in any given section, a partition for which no standards or guidelines on content exist. A document that uses only the section titles prescribed in DoD STD 7935.1-S is in conformance with that standard. Indeed, the authors have audited documents which met standardization requirements in just such a fashion.

FIPS PUB 38 offers essentially the same or greater flexibility as does DoD STD 7935.1-S. But FIPS PUB 38 is a set of guidelines, not a standard. A software documentor implicitly has the option of not following a set of guidelines. The section on flexibility in FIPS PUB 38 merely makes this freedom explicit. However, as stated in FIPS PUB 38, documentation provides information to support the effective management of ADP resources and to facilitate the interchange of information. Adherence to the guidelines enhances the likelihood of achieving these goals. Offering the software documentor the license to completely depart from the guidelines vitiates the program that produced the guidelines.

Documentation standards with flexibility intermediate between the two extremes represented by current standards would be very beneficial. The standards should permit sufficient tailoring of the format and content of the documents to cope with project size and complexity without vitiating the documents. Then the production of all the document types or some minimal subset of them could be required for all projects, with each document type tailored as appropriate for the project being documented.

6. CONCLUSIONS

The software documentation standards considered in this paper were MIL-STD-483, MIL-STD-1679, DoD STD 7935.1-S, and FIPS PUB 38. Based on our experience with this sample, we offer the following conclusions:

1. A tradeoff exists between making each document in a documentation chain stand-alone on the one hand, and maintaining visibility and traceability

using this chain on the other hand. If each document is to be self-contained, visibility and traceability are generally cost-prohibitive to maintain throughout a project's life cycle. If, on the other hand, no overlap among documents exists, then traceability is difficult to establish (i.e., the links in the chain do not interlock and are tangent to one another at best).

2. Again, for purposes of maintaining traceability, a document once produced should be maintained (i.e., kept up-to-date). In the standards in our sample, this document maintainance aspect is not emphasized (and is, at best, implicit).

3. Some of the standards do not take cognizance of the reality that before software is tested, a test plan should first be formulated specifying a testing approach, and then step-by-step procedures detailing this approach should be formulated. FIPS PUB 38 and DoD STD 7935.1-S call for a single pre-test document that combines planning and procedure writing into a concurrent exercise. Since test procedure development is generally an extremely labor intensive activity, performing test planning and procedure writing concurrently can be an extremely profligate activity if the testing approach in the plan does not meet buyer/user approval.

4. Up to a point, a standard for a document should be viewed as a checklist of items to be addressed in the document. Some discretion, however, should be used in applying the checklist. Blind adherence to the checklist can lead to a document that obfuscates rather than clarifies. In the other extreme, keeping section headings but ignoring the basic intent of the items in the checklist makes a mockery of the standard. A documentation standard should be viewed as something that ignites the process of writing a document. It jogs the mind into action, stimulates the creative processes, and bounds them with guidelines that should be followed as reason and project circumstances dictate.

7. REFERENCES

1. "Guidelines for Documentation of Computer Programs and Automated Data Systems," FIPS PUB 38, National Bureau of Standards, February 16, 1976.

2. "Automated Data Systems Documentation Standards," DoD Standard 7935.1-S, Department of Defense, September 13, 1977.

3. Kurihara, T., Redwine, Jr., S.T., and Zaveler, S.A., "Observations on Documentation Standards Revision: FIPS PUB 38 After Four Years," Software Engineering Standards Application Workshop, August 18-20, 1981, San Francisco, CA, IEEE Computer Society Press, Los Alamitos, CA, 1981.

4. "Automated Data System Documentation Standards Manual," DoD Manual 4120.17M, Department of Defense, December 29, 1972.

5. "Programming Documentation Standards and Specifications," NAVCOSSACT Instruction 5230.9, Naval Command Systems Support Activity, August 1, 1966.

6. "Weapon System Software Development," Military Standard MIL-STD-1679 (NAVY), Department of Defense, December 7, 1978.

7. "Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs," Military Standard MIL-STD-483 (USAF), Notice 2, March 21, 1979.

8. Lehman, John H., "How Software Projects Are Really Managed," Datamation, Vol. 25, No. 1, pp. 119-129, January 1979.

Case Studies, Management Guidance and Quality Criteria for Software Docmumentation

by

James N. Orton
Software Documentaton Manager,
ALQ-131 and TWS Projects

During the past decade increasing demands have been put in accelerating fashion upon the software documentation function in the form of (1) the increasing complexity and scope of radar systems software applications, (2) the increasing information requirements of an increasing variety of Government software documentation standards and specifications, and (3) increasing Government attention to enforcing contractor fullfillment of these requirements.

Westinghouse is attempting to meet these demands by developing

(1) Standardized yet non-stultifying approaches to software documentation in acceptance of today's reality of dynamically proliferating and changing Government software standards and specifications;

(2) Automatic documentation tools utilizing word processor and computer systems operating initially apart but eventually in concert;

(3) A data base, built up from accumulating software documentation experience, designed both to develop overall documentation quality criteria and to minimize the documentation startup effort for future projects anticipated to relate in varying degrees to present projects;

(4) Management techniques for putting this documentation process into effect using the most cost-efficient yet ego-sustaining division of labor between the software designer and programmer on the one hand, and the "information specialist" or document format designer on the other.

These ideas will be explained and evaluated in the harsh light of experience over the past year on two Westinghouse radar systems software documentation projects.

Steps Toward a Solution

- Standardization
- Automation
- Data-Based Experience
- "Creator/Documentor Interface" Management

Standardization

- Interpretation of Standards and Specifications
  - MIL-STD-483 (1970 & 1979)
  - MIL-STD-490
  - MIL-STD-1679
  - DID's
- Document Format Specifications

Automation

- Flowchart
- Data Description
  - Internal
  - External
- Set/Use Matrix (Inter-CPC or Inter-Subprogram)
- Data Item Cross Reference
  - Internal
  - External (I/O)
- Document File Management (WP)

Data-Based Experience

- Document Format Specification Files
- Document Files

"Creator/Documentor Interface" Management

- The Creator (S/W Designer, Programmer) is
  - Responsible for Content
  - Indispensable to Documentation Effort Success
- The Documentor ("Information Specialist") is
  - Responsible for Standards/Specs Interpretation
  - Responsible for Format/Information Presentation
  - Part of a Service Organization
    - To S/W Engineering
    - To Project Management
- Both Should Work Together, With Complete Mutual Understanding, From Day One

---

The Real World

- TWS Project
- ALQ-131 Project
- ASPJ/CPMS Project

TWS Project (AF Single-CPU ECM System)

- Documents: CCP, CI, CPM, CSR, DPFS, DS, MM, PS, SDD, TER, CSTP, UM, VDD
- Standards/Specs: Mil-Std-483 (1970), Mil-Std-490, DID's
- Techniques:
  - Document Format Specs
  - Automation of Flowcharts (Annotation Language/Autoflow Adaptation)
  - Word Processor File Library Initiation (DSF's, Docu-Documentation

---

ALQ-131 Project (AF Double-CPU ECM System)

- Documents: CCP, CPS, CPDP, DS(3), IS, PS(3), SR, TR, UM, VDD(3)
- Standards/Specs: Mil-Std-483 (1979), Mil-Std-490, DID's
- Techniques:
  - Document Format Specs
  - Automation of Flowcharts, Data Descriptions (Internal), Set/Use Matrix, Data Item Cross Reference (Internal and External)
  - Word Processor Library Expansion (DFS's, Documents)
  - Management (Centralized)

ASPJ/CPMS Project (Two Navy Multiple-CPU ECM Systems)

Documents: IDS(2), PPS(2), PDS(2), PDD(2), DBDD(2), PPD(2), CPTP, CPTS(2), CPTPr(2), CPTR(2), OM, VDD(2), CPIN Req (2), SDP

- Standards/Specs: Mil-Std-1679, Mil-Std-490, DID's
- Techniques:
  - Document Format Specs
  - Automation of Data Descriptions (Internal), Set/Use Matrix, Data Item Cross Reference (Internal and External)
  - Word Processor Library Expansion (DFS's, Documents)
  - Management (Centralized)

Conclusion

- [Sometimes,] "I'd rather be in Philadelphia."
  - W.C. Fields
- [Nonetheless,] "Well begun is half done."
  - Poor Richard's Almanac
    (or equivalent)

Experiences in Software Standard Selection And Application - A Case History

Thomas L. Hannan and Alice A. Wong

Federal Aviation Administration
Washington, DC

This paper presents the findings of analyses conducted by the Systems Research and Development Service of the Federal Aviation Administration regarding the applicability of existing software standards to the development and implementation of the Air Traffic Control Advanced Computer System. A brief description of system requirements, acquisition methods, and standardization objectives is presented with a description of the standard review activities and resultant findings. Preliminary conclusions based on these findings are described, and the issues pending resolution are identified.

Keywords: Software standards; Selection criteria; Advanced Computer System

1. ADVANCED COMPUTER SYSTEM

The Air Traffic Control (ATC) System is a complex system incorporating navigation, communication, surveillance, and automation facilities, equipments, and personnel. The Advanced Computer System (ACS) will be the major automation component of the ATC system and will provide the ATC specialist the data and information required for planning and control actions. In addition to being complex, the ACS is critical. As the ACS is comprised mainly of software, it is the concern about this software that drives the search for means to insure its integrity and improve its maintainability over the system's life cycle.

1.1 Requirements

ACS software requirements encompass functional, performance, and interface aspects of the system. The software must perform radar data processing, weather data processing, aircraft tracking, flight plan data processing, graphic and tabular presentation, and system-sustaining operations. It must interface with a variety of both long-range and short-range aircraft and weather surveillance equipment, a variety of associated control facilities and user equipment, and a large number of ATC operations and supervisory personnel. The software must be reliable, maintainable, and adaptable to a wide variety of geographic configurations, air traffic density and service profiles, and evolutionary procedural concepts.

1.2 Acquisition

The ACS, including its software, will be procured on a competitive basis involving multiple-year, multiple-contractor solicitations spanning a relatively long time frame. Each of these aspects adds a dimension of complexity to the acquisition. Requirements must be uniformly communicated, provide a common evaluation basis, and facilitate discrimination among offerings. System engineering, verification and validation, development and integration activities must be carefully coordinated. System products must be assimilated into a common operational inventory, technology evolution must be accommodated, and program continuity must be maintained.

1.3 Standardization Objectives

Having defined the requirements and identified the constraints, it remained to formalize the plan and institute the procedures needed to provide assurance that it will be carried out. Procedures were sought that would:

- Assure Adequacy of the software with respect to the specified requirements.
- Contain Risk of the software development efforts within acceptable limits.
- Promote Economy of both the development and operation of the software.
- Assure Interoperability of the software with other system components.
- Promote Adaptability of the software to environmental differences and evolutionary change.

Selection and application of existing software standards were to be accomplished in light of their individual contributions to these objectives.

2. STANDARDS REVIEW ACTIVITIES

2.1 Approach

To formulate the needed set of software standards, a sequence of studies was conducted. First, the areas in which software standards application was documented were surveyed. Second, available standards within these areas were identified and collected. Third, standards were evaluated with respect to the specified objectives; the most appropriate standards were to be selected; and tailorings were to be devised to integrate them into a consistent package.

2.2 Application Areas

Within three major areas (Project Management, Software Development, and Test and Reliability), several categories of standards, were identified. Table 1 indicates the areas, their respective categories, and the objectives addressed by each category of standard.

3. FINDINGS

Existing standards do not comprise a complete, consistent set of procedures which satisfy the needs of the ACS effort. Three attributes of the standard set identified as being most appropriate are worthy of note: (1) Varying Objectives--attributable to originating source; (2) Varying Level of Specificity--attributable to a lack of elementary principles, and (3) Unaddressed Major Areas--attributable to the lack of concurrence among current practitioners.

3.1 Varying Objectives

The National Bureau of Standards, the American National Standards Institute, and the Department of Defense constitute the primary sources of standards identified. Each source addresses a different audience from a different perspective with a different terminology. The result of attempting to combine them is to incur a significant "normalization" effort.

3.2 Varying Levels of Specificity

Four types of standard documents are identifiable; definitions, exposition, recommendations, and requirements. Firm definitions or expositions of elementary principles are noteworthy by their absence. Recommendations and requirements exhibit a tendency to cluster in the more "mechanical" categories, and vary widely in level of detail. Requirements appear to be based less on demonstrated efficacy than on personal preference.

STANDARD	OBJECTIVE				
	Adequacy	Risk	Economy	Interoperability	Adaptability
PROJECT MANAGEMENT					
Life Cycle Definition	X	X	X		
Configuration Control	X		X		X
Documentation	X		X		X
Human Factors	X			X	X
Simulation	X				X
Miscellaneous				X	X
SOFTWARE DEVELOPMENT					
Languages		X	X		X
Programming		X	X		X
Data Bases		X	X		X
Computer Communication				X	
Media	X			X	
Hardware/Software Interface	X		X	X	
TEST AND RELIABILITY					
Quality Assurance	X	X			
Reliability/Maintainability		X	X		
Test/Test Documentation	X	X			
Terminology					X

TABLE 1: Objectives of Software Standards

3.3 Unaddressed Major Areas

Adequate standards for representing software requirements, software designs, and actual code were not identified. No guide to the applicability of a given representation to a given application was identified, and no standard for a given representation appeared to have universal agreement. Reliability and maintainability standards were not identified although candidate techniques exist.

4. CONCLUSIONS

A complete, consistent set of standards is needed, but is currently not available. To use an incomplete set would be to risk the value of those that were used, and to use an inconsistent set would be to incur unnecessary costs at best. At first glance, it would appear that a monumental effort would be required to resolve the problem.

Upon further thought, however, it becomes clear that creation of a total set of standards covering everything from general life cycle definition to detailed coding standards is not what is needed. Careful examination reveals two distinct classes of standards; standards with a "capital S", and standards with a "small s".

- "Standards." These define mandatory requirements based on universally accepted principles. They should be limited to the specification of "what" should be done rather than "how" it should be done. They should address product attributes of form, fit, or function. They should be durable. They are in the domain of the purchaser. Currently, they are "meta-standards" that require compliance with a specific "standard" (small s) to be negotiated.

- "standards." These define specific alternate instances which satisfy the requirements of "Standards" (capital S). They should specify in detail "how" a requirement is to be met. They should address process methods and procedures. They should reflect the most appropriate technology. They are in the domain of the seller. Currently, they are recommended practices generally integral to a development organization's preferred way of doing business.

This view is consistent with generally accepted specification practices and also appears in keeping with the intent of the Federal Information Processing Standards produced to date.

From the purchaser's perspective, then, what is needed is a complete and consistent set of "Standards" that cover the generic needs of system development aspects such as quality assurance, documentation, and training. It would seem to be the function of the competitive marketplace to produce successively better "standards" that satisfy these needs.

In the preparation of each type of standard, care should be taken not to infringe on the realm of the other type. "Standards" should not dictate specific techniques at the expense of innovation. Similarly, "standards" should not be promoted as requirements for the same reason. There is a place for a "Standard" for design documentation and there is a place for a "standard" for program design language; they should be kept in their respective places.

Finally, in the preparation of standards, especially "Standards", efforts should be expended in alleviating the deficiencies already noted. Parochial sources of standards should agree on common objectives; they should at least agree on a common terminology. Standards should be based on principles; empirical if not elementary. Standards should have the concurrence of those affected; buyers and sellers alike. We cannot, each of us, continue to pursue our own individual paths and hope to successfully meet the challenges of the future.

REFERENCES

Hecht, H., ATC Computer Replacement Program Software Standards Guidebook, U.S. Dept. of Transportation, Federal Aviation Administration, Systems Research and Development Service. Volumes 1, 2, and 3, Washington, D. C., February 1981.

Proceedings, Software Engineering Standards Application Workshop. IEEE Computer Society. IEEE Cat. No. 81 CH1633-7, San Francisco, August 19-20, 1981.

SESSION A: Applying Documentation Standards

Summary of Findings and Recommendations

R. J. Gavin
Federal Deposit Insurance Corporation

The papers presented during this session highlighted various problems encountered by software groups both inside and outside the Federal government as they have attempted to cope with the array of existing software documentation standards.

Documentation preparation should be treated as a continuing effort, evolving from the preliminary draft through changes and reviews to software delivery, with subsequent updates indicated by user feedback. If the documentation develops as the code develops, by the time the code is completed the documentation will be current.

A. Existing Standards

Existing guidelines provide some good "how-to" approaches to the preparation of basic software documentation. The participants in this session indicated areas in which these approaches could be improved:

1. Many expressed a general lack of awareness as to what software guidelines are currently available. Therefore, a significant segment of the users are reinventing the wheel instead of utilizing software guidelines which are already in existence.

2. FIPS PUB 38 guidelines discuss ten basic documents. There is, however, significant redundancy between the various documents in a set. This redundancy could be reduced while still retaining the necessary informational content for project auditing, analysis, or update.

3. The current software guidelines do not adequately address system security or interactive applications. The emphasis is on batch systems with discrete input and output.

4. Participants in this session expressed the desire for some guidance regarding the relevance and appropriateness of the different documents to widely divergent types of applications software.

B. New Standards

A consensus developed that new guidelines should be created to address the areas of:

1. All phases of software system life cycles.

2. System maintainability, reliability, security, and testing.

C. Analysis and Research

Speakers and attendees discussed several areas for additional research and development;

1. The implementation of documentation for the automated office.

2. The rapid advance of user-developed-and-implemented software.

3. Guidelines for testing and verification of software documentation.

4. Use of new technology and automated support tools for documentation preparation and maintenance.

D. Conclusions

The users represented in this session have applied existing FIPS software documentation guidelines with mixed results over uniformly long periods of time. They expressed keen interest in the next steps in the development of software documentation standards and are much aware of the dangers inherent in the spread of undocumented software.

Session B: Documenting for Operation and Maintenance

Introduction

Charles L. Gerhardt

U.S. Department of Agriculture

The National Bureau of Standards describes the software life cycle as consisting of three general phases. During the Initiation phase, the objective and general definition of the requirements for the software are established. During the Development phase, the software is maintained, evaluated and changed as additional requirements are identified. These phases and their definitions are arbitrary but they are adequate for discussing the process of software development.

FIPS Publications 64 and 38 present in detail, documentation content guidelines for the Initiation and Development phases respectively. There are no documentation content guidelines for the Operation phase. Is there a need for documentation guidance for this phase? The General Accounting Office report entitled Federal Agencies' Maintenance of Computer Programs: Expensive and Undermanaged (ASMD-81-25, February 26, 1981) presents the results of a survey of over 400 governmental data processing installations. This survey indicated over 50% of software maintenance results from modifications or enhancements required to make the software perform more in the user function. Federal agencies spend millions of dollars annually on computer software maintenance and it is during the Operation phase that sound management, including documentation of software modifications and enhancements, must continue.

The papers presented in this session describe how FIPS documentation guidelines have been adapted in a governmental agency; present an approach used to document modifications of a commercial software product; point out the unique documentation requirements of on-line, real-time systems; and propose specific documentation content guidelines for the Operation phase of the software life cycle.

NBS FIPS Software Documentation Workshop,
March 3, 1982

The Development and Implementation of Uniform ADP Documentation Standards at FAA

Harvey P. Kaplan

Department of Transportation
Federal Aviation Administration
Office of Management Systems

This paper chronicles the experiences of the Federal Aviation Administration (FAA) in tailoring uniform documentation standards to the guidelines contained in FIPS PUB 38. The end product which emerged from this effort was an agency directive which defines requirements for technical documents produced during the development of all approved automated data systems. The directive consists of twenty three specific document content standards defined as documentation elements. These documentation elements are further arranged into documentation categories which are guidelines for packaging the elements. This paper also describes the approach employed in developing the standards, significant benefits accruing from the use of the standards, and concludes with a summary of conclusions and additional needs which were a direct result of uniform documentation standards.

Keywords: Documentation categories; Documentation elements; Uniform documentation standards; User guide documentation standards; User involvement.

I. INTRODUCTION

The Federal Aviation Administration (FAA) has long recognized the vital importance of sound documentation practices in the operation of it's data processing facilities. The issuance of FIPS PUB 38 in 1976 coupled with major decisions regarding agency software and hardware systems heightened the requirement for uniform documentation standards. The FAA has been a long standing advocate of FIPS PUBs in conjunction with its own ADP Standards Program. However, analysis of FIPS PUB 38 quickly revealed that a further translation was required in order to introduce a meaningful document to the FAA ADP community. Accordingly, the thrust of this effort centered on tailoring FIPS PUB 38 to the practical needs of FAA, while still retaining its structure and intent.

The purpose of this paper is to describe FAA's experiences in developing uniform documentation standards; highlighting major benefits of documentation standards; and concluding with findings, advantages, and recommendations.

II. SUMMARY RECOMMENDATIONS

A comprehensive set of documentation standards is essential to the management of any ADP system. System documentation provides the chief means by which personnel responsible for the design, implementation, use, and operation of a system communicate with each other.

The benefits which accrue to a large, geographically dispersed organization such as the FAA are significant in terms of efficient use of hardware, software, and personnel resources. Documentation standards enable document orginators to prepare their material according to a preset logical format. The documents produced will be familiar to others in the organization and designed to meet common needs.

The quality of system software and applications programs is also enhanced by standardized documentation. Applications which are operated at multiple facilities become more transferable from one facility to another. Duplication of effort among facilities is thereby minimized since standard documentation promotes a general awareness of the applications available throughout the organization.

Finally, good ADP documentation practices facilitate personnel training since both the standards and resultant documents provide an excellent source of training material. The resulting shorter learning times tend to minimize the impact of transfers or other ADP personnel turnovers.

III. TECHNICAL DISCUSSION

The success of the overall effort hinged on gaining acceptance and receiving cooperation from key personnel dispersed

throughout FAA's 14 ADP facilities. In order to achieve this goal every effort was made to seek involvement and participation from personnel who would ultimately be the users of the documentation standards or be responsible for their completion. Both formal as well as informal lines of communication were established throughout the duration of the project.

Since most facilities had some form of existing documentation, one of the first tasks was to complete an in-depth review of the current standards and adopt those that were appropriate for consideration. One of the significant challenges during this phase of the project was to develop standards which strike the proper balance between flexibility and uniformity. The standards also had to reflect a sensitivity to the requirements of FAA's regional offices so their responsibilities would not be subordinated to unnecessarily rigid controls.

It has been FAA's experience that standards are not self-implementing, but require a participative role on the part of the users. Therefore, no effort was spared to maximize the communication between all parties involved in the effort. Formal coordination was utilized at appropriate stages of project development. Where significant conflicts arose, compromises were reached to the satisfaction of affected offices. At all times an atmosphere of open and candid communications was maintained.

The product which emerged embodied the concepts established during the project approach phase. Thus, the nature and extent of documentation required during each phase of system development vary, depending on the scope, complexity, and type of system being documented. Four levels of documentation were established to accommodate these varying requirements. This concept permits flexibility in documentation requirements, thus eliminating burdensome documentation for simple, less complex systems.

Document content standards defined as documentation elements provide specific requirements concerning the documentation of an automated data system. The FAA standards prescribe up to twenty three of the following documentation elements:

1. Documentation Check List
2. System Description
3. Program Description
4. Functional Flowchart
5. Process Flowchart
6. Program Flowchart
7. File/Data Base Decription
8. Input Document Definition
9. End Product Definition
10. Data Element Description
11. Data Grid
12. Center/Region Interface
13. Job Control
14. System Run Information
15. Data Collection and Preparation
16. Peripheral Process Guide
17. Distribution Guide
18. User Job Initiation Procedures
19. Output Review and Resubmission
20. Remote Processing
21. System Test Procedure
22. Evaluation of System Test Results
23. Glossary of Terms

The documentation elements are further arranged into the following documentation categories:

1. System Design Specification
2. System Documentation
3. Program Documentation
4. Computer Operating Documentation
5. User's Guide
6. System Test Plan
7. Data Element Documentation
8. Distribution Guide

Concurrent with distribution of the approved directive, a videotape training presentation was prepared which enhanced understanding of the documentation standards.

IV. CONCLUSIONS

The FAA has reaped numerous benefits from utilization of uniform documentation standards. Most prominent of these include an increased awareness of software sharing, eliminating duplication of effort, minimizing work disruption created by personnel turnover, providing an excellent source of training material, and contributing to the overall efficiency of it's data processing facilities. In addition, several favorable by-products emerged.

Faced with austere staff levels, the FAA has placed increasing reliance on contractual support. The documentation standard directive has proved to be a useful measure in determining contractually produced documentation. FAA contracting officers normally require that the directive be cited in the statement of work for software

development efforts. Thus, it has provided an excellent source of documentation requirements for both contractor and FAA, at the negotiation stage, thereby eliminating any ensuing misunderstandings. The reaction from contractors has been overwhelmingly positive. Several have even suggested they intend to adopt FAA documentation requirements for their own internally developed systems.

A post evaluation of the uniform documentation standards led to the development of user guide documentation standards. Though addressed in the former document, increased emphasis on distributed data processing systems and growing reliance on user interaction dictated the need for a separate directive. The user guide standard represents a logical extension of the earlier effort. The documentation contained in this directive is geared to support non-ADP personnel who interact with ADP systems.

In retrospect, if there is a single dominant quality which characterized the success of producing uniform documentation standards, it was strong communications. By their very nature, standards are not appealing to the recipient. However, when it is demonstrated that an honest and sincere effort is being made to incorporate each organization's thoughts in developing standards which can make their job more efficient and orderly, a productive relationship is established. This rapport can be a source of meaningful achievements and should be regarded as a valuable asset.

SUPPLEMENTAL DOCUMENTATION OF MODIFICATIONS TO SOFTWARE PRODUCTS ON SMALL TO MEDIUM SIZED SYSTEMS

BY HENRY A. LEWIS

DCD COMPANY/DIVISION OF BORG ENTERPRISES

Supplemental Documentation of Modifications to Software Products on Small to Medium Sized Systems is designed to provide a simple structure for analyzing proposed modifications, documenting development, and archiving pertinent project documents. This method is supplemental to existing documentation procedures. The system approach is to keep the method basic and straightforward while highlighting such historicaly troublesome areas as development plans, future referencing, and audit concerns.

SOFTWARE PRODUCTS; SUPPLEMENTAL DOCUMENTATION

1. INTRODUCTION

The following paper describes the method of documenting enhancements to established software systems. The purpose of this method is to provide a consistent pattern of analyzing proposed modifications, documenting development progress, and archiving pertinent project documents. The methodology is designed to provide a simple structure for documenting the development and implementation of enhancements to an existing system. The system is assumed to be fully documented in its own right, whether a privately developed program or a marketed software product. This method is supplemental to existing documentation for such programs and products. Although enhancements are viewed through predefined phases or categories, necessary attention is given to project development, archival documentation, and company audit concerns. In terms of new systems implementation, it is conceivable that this straightforward approach will find applicability for shops not having formal implementation and documentation procedures as yet. In such cases, the shop will find it necessary to elaborate on the basic method to form a company's complete software documentation procedure.

2. DESIGN CONSIDERATIONS

The Methodology for Supplemental Documentation of Modifications to Software Products on Small to Medium Sized Systems (SDMS) developed as the need to modify formal software products became necessary. Such products have their own documentation. However, no method was available for developing and implementing modifications to the products. The lack of such an approach identified an urgent need. SDMS was designed for the ongoing software environment. It is intended to document modifications to existing systems. The approach is to keep the method basic and straightforward while highlighting such historically troublesome areas as project development plan, future referencing, and audit (control) concerns. This method is not intended to replace established documentation procedures, but is supplemental in nature. It is designed to complement current procedures. However, where no procedures exist, this approach (with certain additional mandatory inclusions) offers a viable framework for custom system implementation and documentation.

Software products are designed to appeal to a sufficiently large enough market to justify their original development costs. Consequently, it is their nature to be somewhat broad having a generalized solution to the problem they address. A company that wishes to implement such software products must either keep their current systems in place to the point where the product takes control, or modify their existing systems, or modify the product itself to accommodate the necessary requirements of the company. During these periods of review, close examination is often given to the company's existing system with an eye toward improvement. Where company changes are needed, company procedures are modified. Where systems analysis identifies no necessary changes, the proposed product is evaluated for applicability to present methods. If the product is sufficiently different from current procedure, it is recommended that modifications to the product remain close to the original product design. The final form of the system undoubtedly reflects compromises from both sides. But the rationale is that any modification to the software product today is a maintenance task tomorrow.

SDMS provides not so much a concise method as a comfortable framework from which to analyze a proposed need for product modification, describe how, what, and when the change will be, and retain permanent records of the developments once completed. The simple structure offers ample latitude for existing procedures and individual initiative while providing a consistent development procedure.

3. PROCESS DESCRIPTION

The user recommends a change to MIS by completing the REQUEST section of the Project Implementation form (PI) (Exhibit A). This section describes in general terms what the enhancement consists of. PI is submitted to MIS development for consideration. Requests are assigned consecutive project numbers. An initial review of the request is conducted by MIS to determine reasonableness.

A

PROJECT IMPLEMENTATION

PI

PROJECT NO _____

REVISION _____

REQUEST

REQUESTED BY _____ DATE _____

DESCRIPTION OF CHANGE

REASON FOR CHANGE

ACTION

ORIGINATED BY _____ DATE _____

PROJECT NO _____ REVISION _____

DESCRIPTION OF MODIFICATION

IMPACT ON CURRENT ENVIRONMENT

IMPACT ON COSTS & SCHEDULES

REVIEWED BY _____ DATE _____

ACCEPT ( ) REJECT ( ) REASON FOR REJECTION _____

CONTROL

FORM COMPLETION ___

INDEX ENTRY ___

COMPUTER LISTINGS ___

ATTACHED DOCUMENTS ___

PACKAGE TRANSFER ___

VERIFICATION ___

DEVELOPMENT APPROVAL

IMPLEMENTATION APPROVAL

DEVELOPMENT

ANALYSIS PRELIMINARY REVIEW ASSIGNED TO _____

FEASIBILITY

DESIGN INTERIM REVIEW ASSIGNED TO _____

PROGRAMMING

EVALUATION TECHNICAL REVIEW ASSIGNED TO _____

IMPLEMENTATION

POST-IMPLEMENTATION AUDIT

REFERENCE

ATTACHMENTS

MIS responses from this initial review are recorded in the ACTION section of the PI. The project is approved for analysis and feasibility, or disapproved with sufficient reason.

SDMS consists of a project-oriented documentation base. Each request is viewed as a unique modification except in cases where new enhancements add to or obsolete existing modifications. In such cases, project number revision levels are incremented, or are signified inactive, respectively. SDMS control derives from the PI. This central, uniform document summarizes the development process, and provides basis for maintaining a standard set of documentation for all enhancements. This document highlights the development plan and cross-references additional documents. Project implementations are assisted by this consistent approach to the development process through the common PI reporting format.

Project development is planned by MIS. PI allows for describing a project in seven categories. Based on the size of the modification and/or standard company procedures, these categories are given more or less attention as the situation dictates. The development process is defined by these phases. Existing company development procedures may require additional, more definitive documentation. These phases represent a basic logical progression in development thinking.

3.1 Analysis

User requirements are defined. Interviews are conducted to determine project requirements. A preliminary review is held with users to discuss findings. Project definition is reworked as required.

3.2 Feasibility

Reasonability of project per operational and/or economic considerations is examined. Pertinent information is gathered and analyzed. The impact of modification per operation and/or cost concerns is documented.

3.3 Design

Design specifications are established. Project specifications are provided with necessary support documents. An interim review is held with programming support to clarify design specifications.

3.4 Programming

Modifications are coded and tested. Programs and procedures are coded per design specifications. Flowcharts and logic narratives are included as required.

3.5 Evaluation

Project results are evaluated. A technical review of project is held with programming support. Project goals are compared to results. An evaluation is held with a user review board. Attention focuses on project objectives. Review board either recommends implementation, resubmits project to design and programming phases, or decides against implementation.

3.6 Implementation

Implementation plans are established. A firm implementation plan is developed. Plan is presented to users for approval. Implementation schedule is reworked as required.

3.7 Postimplementation Audit

A postimplementation audit is conducted to determine the modification's overall effectiveness. Results are published to management.

4. PROCESS EVENTS

This section summarizes a typical series of events comprising the development process.

4.1 Project Action/Preliminary Review

The PI is submitted to MIS development. Development assigns a project number and the ACTION section is completed. This includes the reason, a description, and the impact of the modification on current environment. Additional support documents such as written requests, impact statements, etc., are referenced in the reference section and attached to the form.

4.2 Development Approval/Project Plan

The project is approved for development by MIS management. A plan is established including review and completion dates, assignments, and a brief description of each phase of activity. The PI works as a reference document during analysis, feasibility, and design of the project. Supporting documents generated during these phases are referenced on the PI and attached.

4.3 Design/Interim Review

An interim review with programming support is conducted during the design phase to clarify project goals and to identify conflicts between planned design and coding limitations.

4.4 Programming/Technical Review

Development releases the PI to programming support for coding and testing. Mandatory contributions from programming activities are computer listings of pertinent procedures and program compilations. These are referenced on the PI and attached. Programming support releases the PI to development during evaluation. Overall project goals and test results are reviewed by analysts with programming support from a technical perspective.

4.5 User Evaluation/Implementation

Development presents project results to the review board for evaluation. The review board is comprised of user groups and MIS management. Implementation follows a favorable review and appropriate approvals.

4.6 Document Package Creation/Backup

Projects are recorded in one of two indices—one indexing nonimplemented, the other implemented (Exhibit B).

The attachment of all supporting documents referenced on the PI is verified. The originals and duplicates of the PI and all attachments are brought together as document package and backup, respectively. The original is retained in MIS local files. The duplicate is archived off-site. Proper assemblage of the packages is verified by a second party.

4.7 Postimplementation Audit

Several months after implementation, an audit of the modification is done. Project goals are reviewed for the effectiveness of modifications, and any subseqent problems are examined and documented. Audit results are included in the document package, and published to management.

5. PROJECT IMPLEMENTATION FORM

5.1 Request Section

The REQUEST section identifies the individual requesting a change or modification and the date of the request. A description of the change from the user's perspective is given, and a statement summarizing the need for change.

5.2 Action Section

The ACTION section identifies the individual originating the project and the date of the origination. MIS personnel are responsible for project number generation and control. A project number and any subsequent revision is assigned. A description of the modification views the change from a technical perspective. Effects on the current environment are highlighted. The request is examined to determine what specific library elements and data files would be altered by enhancements. The degree of the change is given in terms of lines of code changed or added. Dynamic procedural impacts are also described. A formal impact statement may be required. If so, it is identified in the REFERENCE section. Any impact on costs and schedules is described. The individual reviewing the request is identified and the date of the review. The review either accepts the request for development, or rejects it. If rejected, sufficient reason is given.

5.3 Development Section

The DEVELOPMENT section describes the goals of each particular phase or category of the development process. Tentative completion dates and specific deliverables are given for each. Responsibility for the three mandatory reviews is assigned. Possible inclusions at each point might be:

Analysis	-	Statement of project scope, system surveys, and analyses of project requirements.
Feasibility	-	Feasibility studies.
Design	-	System schematics, job stream descriptions, design narratives, and file layouts.
Programming	-	Flowcharts, and logic narratives.
Evaluation	-	User evaluation reports.

Implementation - Detailed implementation schedule.

Postimplementation - Audit report.

5.4 Control Section

The CONTROL section identifies proper PI completion, respective Project Number Index entry, mandatory inclusion of computer compilations and procedure listings, attachment of all referenced documents, proper assemblage of the document package, and the reverification of the foregoing by a second party. Verification is signified by initialing. Development and implementation are each approved by MIS management.

5.5 Reference Section

The REFERENCE section identifies by name each document referenced and attached throughout the development process to form the document package.

6. PROJECT NUMBER INDEX FORM

The Project Number Index includes an entry for each project number/revision level occurrence. The date of the entry, plus libraries, elements, element types, and number of lines changed or added is given. The individual logging the entry is identified along with a brief description of the modification. It is recommended that the Index be stored on disk, and that retrieval consist of two or three reversing formats as the MIS function has need.

7. GENERAL CONSIDERATIONS

7.1 Document Package

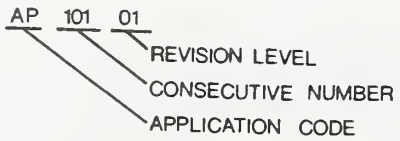
Control of modification documentation derives from the PI. Pertinent supporting documents are logged in the REFERENCE section of the PI. This group of documents represents the Document Package. Contents vary from project to project. However, a certain number of mandatory inclusions are necessary. They include program compilations and procedure listings. It is recommended that design narratives describing basic system flow be provided for anything beyond minor modifications. The comprehensiveness of the Document Package is determined by the impact of enhancements to the existing environment. All attachments are identified by a PROJECT DOCUMENT stamp.

PROJECT DOCUMENT
PROJECT NO_____ REV _____
DATE _____ TOTAL PGS_____
SUBMITTED BY_____

Shops using this methodology as their documentation base should require as mandatory inclusions such things as analysis and design narratives, feasibility studies, system schematics, file layouts, logic narratives, and detailed implementation schedules.

7.2 Composition of Project Number

Some thought should be given to the method of assigning project numbers. It is recommended that the composition of the project number be comprised of three parts. The first portion to represent the application affected by the modification. The second portion to represent sequential numbering of projects, and the third portion to represent subsequent revisions to projects.



The composite project number of the above example is AP10101. This allows for grouping by application and permits distinct filing for each revision level.

7.3 PARALLELING ACTIVITIES

The development process requires the timely participation of all involved. The PARALLELING STRUCTURE/LEVEL/PHASE diagram (Exhibit C) illustrates this process of coordination between each function and phase, and how they coincide with the SDMS methodology.

8. CONCLUSION

The SDMS methodology adopts a simple, straightforward approach to the development of modifications for software products. It has potential as the basis for complete system documentation. However, its primary thrust is as a supplement to existing documentation. Design concerns consist of maintaining method simplicity and a supplemental nature to the scheme. Practical problems occurring during the implementation of changes are reviewed. A general framework rather than a concise formula is established for identifying control points within the development process.

The process is divided into seven categories: Analysis, Feasibility, Design, Programming, Evaluation, Implementation, and Postimplementation Audit. Each category is described and a typical series of development process events is presented. The movement of forms and supporting documents plus various approvals and reviews is described. The use of two forms is discussed in detail. The Project Implementation (PI) form serves as the control document for the whole of SDMS. The Project Number Index simply indexes the many projects.

Several general considerations are discussed including the formulation of a Document Package, a suggested composition for the project number, and a diagram illustrating dynamics of development process interplay.

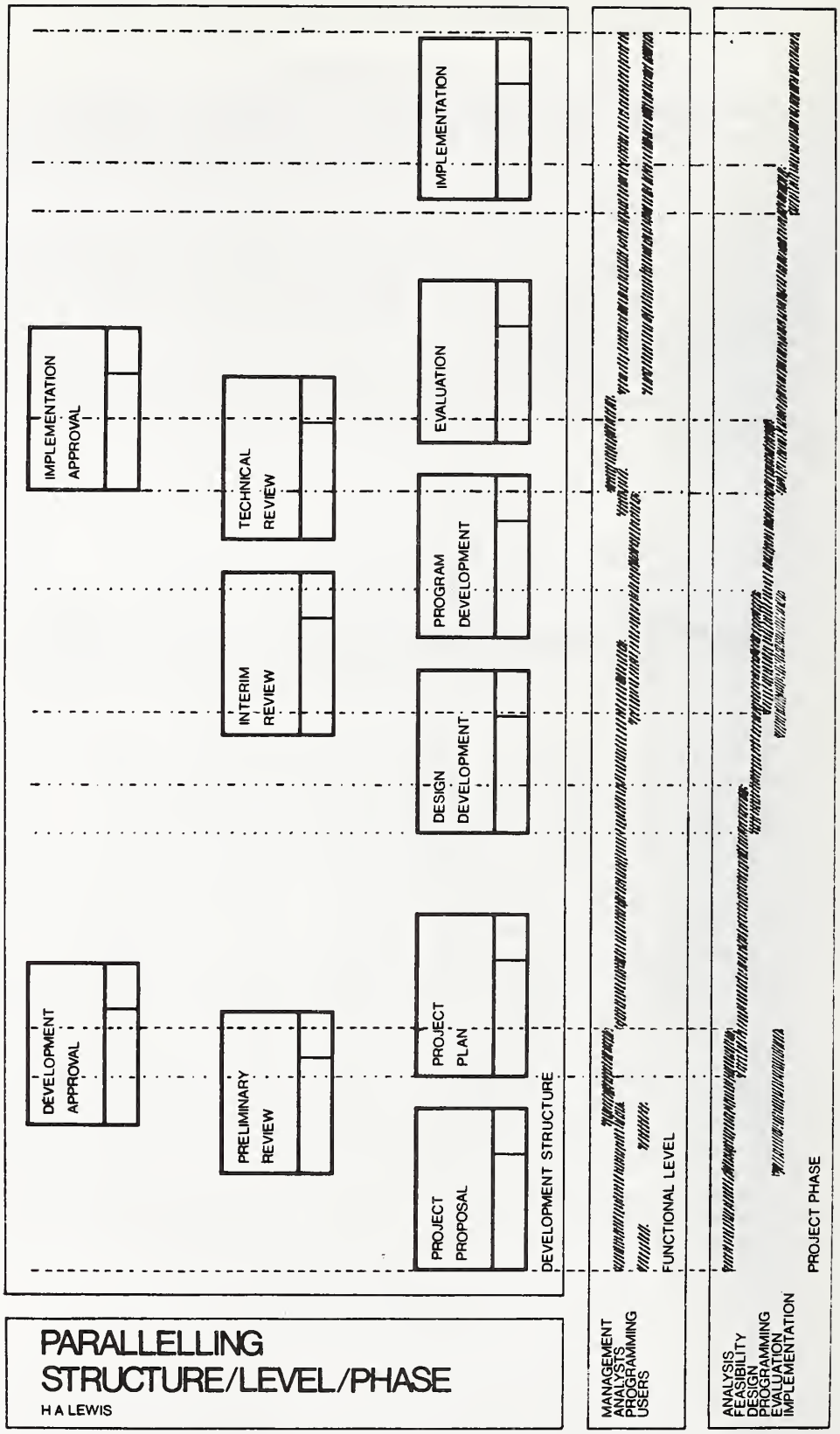
B

PROJECT NUMBER INDEX INDEX DATE ____________

PROJECT NO	REV	DATE	LIBRARY	ELEMENT 1	TY 1	LN 1	E. 2	2	2	E. 3	3	3	LOGGED BY	DESCRIPTION

saa

C


PARALLELLING
STRUCTURE/LEVEL/PHASE
H A LEWIS
DEVELOPMENT STRUCTURE
PROJECT PROPOSAL
DEVELOPMENT APPROVAL
PRELIMINARY REVIEW
PROJECT PLAN
DESIGN DEVELOPMENT
INTERIM REVIEW
PROGRAM DEVELOPMENT
TECHNICAL REVIEW
EVALUATION
IMPLEMENTATION APPROVAL
IMPLEMENTATION
FUNCTIONAL LEVEL
MANAGEMENT
ANALYSTS
PROGRAMMING
USERS
PROJECT PHASE
ANALYSIS
FEASIBILITY
DESIGN
PROGRAMMING
EVALUATION
IMPLEMENTATION

Operations Documentation Standards -- Online, Real-time Versus Offline, Batch

Deborah A. Harman

OCLC Online Computer Library Center, Inc.
Dublin, Ohio 43017

The FIPS 38 content guidelines for the Operations Manual should include or expand four topics important to real-time processing: hardware configuration; start-up, shutdown, and performance monitoring procedures; error messages; and non-routine procedures.

Keywords: Documentation; Operations manual; Real-time system.

1. INTRODUCTION

The operations documentation developed for an offline, batch system must be somewhat different from that developed for an online, real-time system. Offline, batch processing, besides being the simplest and cheapest form of processing, also requires far less operator training and support than does real-time processing. Real-time systems are installed to satisfy users' needs for immediately updated information.[1] While some errors and reruns can be accommodated in a batch environment, the users' total dependence on the system in a real-time environment makes any service disruption intolerable.[2] Operators must be prepared, both through training and documentation, to respond immediately to real-time processing problems.

This need for operator responsiveness to ensure real-time system availability is not adequately addressed in FIPS 38. Specifically, the content guidelines for the Operations Manual should be revised to include or expand the following topics:

- Hardware configuration
- Start-up, shutdown, and performance monitoring procedures
- Error messages
- Non-routine procedures

2. OCLC'S OPERATIONS DOCUMENTATION

2.1 Batch and Online System Documentation at OCLC

Some of the same weaknesses found in the FIPS 38 Guidelines have been revealed in the documentation prepared for the operators of the OCLC Online System. Our experience in evaluating and revising this documentation is offered here in the hope that it may save others, creating documentation for an interactive system, some time and expense.

OCLC provides bibliographic resource sharing and support services to over 4,400 terminals in 6,000 libraries throughout the United States, Canada, Mexico, and Great Britain. Through the OCLC telecommunications network, libraries can access and update a data base consisting of eight million catalog records. Shared use of OCLC's online data base relieves libraries of the necessity to perform many manual, labor-intensive tasks and helps libraries to reduce rising costs and to improve services. Because of the importance of system availability and response time to its member libraries, OCLC carefully monitors the performance of the Online System.

Not surprisingly, OCLC places far greater emphasis on the documentation prepared for its online processing than on that prepared for its batch operations. Not that OCLC's batch operations are trivial; each week, the Production System generates, among other things, 2.5 million catalog cards for shipment to libraries. However, standards for batch operating procedures are only now being developed, and the batch documentation itself exists in a variety of forms (including notes and memoranda). An interdivisional effort is underway to address batch documentation problems. In contrast, OCLC has prepared, for the Online System operators, an enormous volume of material -- the Systems Operations Reference Manual (SORM) -- 450 pages in length.

2.2 The Operations Manual Questionnaire

The Systems Operations Reference Manual, originally released in 1979, is revised quarterly by the Documentation Department of OCLC to reflect software, hardware, and procedural changes affecting the Online System. Late last summer, wondering whether our revision efforts might be better directed, we prepared a questionnaire posing for evaluation 32 specific statements about the manual's completeness, accuracy, content, organization, style, and format. This questionnaire was distributed to 70 holders of the SORM.

It was encouraging to learn from the responses to the questionnaire that the holders of the SORM were actually using it; in fact, most of the computer operators indicated that they were referring to the manual on a daily basis. However, the evaluative responses to the questionnaire were less than enthusiastic. Initially, it seemed impossible to form a revision plan to resolve all the problems noted by the respondents to the questionnaire. Finally, by selecting only statements to which over half of the respondents had expressed strong agreement or disagreement, we were able to identify those aspects of the manual requiring immediate attention. Unfortunately, the major problem area was the material itself; respondents indicated that the information in the SORM was, in some cases, outdated and, in general, incomplete. Specifically, they suggested the following topics for inclusion or expansion in the manual:

- Overview of the OCLC Online System
- Online System Responsibilities by Position and Organizational Unit
- Hardware Description and Operation
- Recovery Procedures
- Problem Detection and Resolution
- Emergency Priorities
- Safety and Security Procedures.

2.3 Addressing Problems in OCLC's Operations Manual

The problem of outdated information, though serious, is not difficult to resolve. Auditing current practices in the Data Center against documented procedures should identify specific items requiring correction in the manual. Better control procedures, to verify that revision material is prepared as system changes are implemented, will also help to keep the SORM's content current.

However, the problem of incompleteness will be costly to resolve, since eventually it will necessitate the total reprinting of the manual. Among the topics suggested for expansion, only the last two -- Emergency Priorities and Safety and Security Procedures -- are not included at all in the current manual. Although these topics are covered in the corporate standards manual, they do need to be detailed specifically in terms of Data Center operations, and will, therefore, be added to an administrative portion of the manual.

The remaining topics require more emphasis and detail than presently offered. This will necessitate not only adding new material, but also reorganizing existing information. For example, all error messages are now listed with their page references at the end of the SORM, but the explanations and appropriate operator responses for the messages are scattered throughout the text of the manual. A computer operator may need to flip from one section of the manual to another to understand clearly what a given error message means and what he should do about it. An obvious first step in revising the manual is to place all the information concerning the messages in a single appendix. Similarly, recovery procedures and instructions for detecting and resolving problems are located in several sections of the manual. Separate, clearly defined sections -- Performance Monitoring Procedures and Restart/Recovery Procedures -- will be added.

3. CONCLUSIONS

3.1 OCLC's Documentation and FIPS 38

The needs expressed by OCLC's Online Operations staff are typical of the needs of all operators working in a real-time system environment. The FIPS 38 content guidelines, to some extent, imply the inclusion of such topics as those listed by the users of OCLC's manual. However, these topics must be identified separately and highlighted as part of the Operations Manual to ensure the availability of this information to operators when they need it. Careful consideration of operators' needs in advance of publication will reduce the need for major subsequent revisions.

3.2 Hardware Configuration

FIPS 38 should be revised to specify a discussion of hardware as part of the Operations Manual. Possibly, in a batch environment, no detailed discussion of hardware is necessary; where the same or similar equipment is likely to be used for all processing, vendor documentation may be enough. However, online systems typically have multi-vendor hardware configurations, and, when problems arise, response time may well be dependent on the operator's understanding of the equipment. In an online, real-time system, there is also no batch job ticket to indicate to the operator which machines are used for running any given program. Therefore, Hardware Configuration should be added to the OVERVIEW Section, as paragraph 2.1, with the subsequent topics renumbered accordingly, e.g., 2.2 Software Organization. The discussion factors for the 2.1 Hardware Configuration paragraph need not be very different from those suggested for the discussion of Equipment in the other document types. The following guideline for discussion factors might provide useful background information to operators:

> Identify the equipment required for the operation of the system. Describe any significant operating features, such as the function and use of lights, switches, and keys. Relate the hardware to specific processing functions or applications. When appropriate, provide a diagram showing functional relationships between mainframes, communications controllers, and peripheral devices.

3.3 Start-up, Shutdown, and Performance Monitoring Procedures

FIPS 38 should also be revised to specify system start-up, shutdown, and performance monitoring procedures. These procedures should be included as separate paragraphs for discussion in Section 3, DESCRIPTION OF RUNS. Start-up, shutdown, and monitoring procedures can be viewed as types of runs, but in an online, real-time environment, they may also be the only clearly identifiable routine jobs, and thus merit special attention. The complex sequence of operator actions required for system start-up and shutdown and the importance of timely problem detection also suggest that specific mention (as paragraphs 3.1, 3.6, and 3.7) is appropriate in the content guidelines.

3.4 Error Messages

A list of all error messages should be added to the end of Section 3. This list could be organized in any way appropriate for the particular system, but should include such information as:

- an explanation of the error message;
- identification of the process or program generating it;
- an indication of its severity;
- action or response required of operations personnel.

Consolidating this information toward the end of the Operations Manual would make it more accessible to operations personnel and could result in less downtime. The list creates some redundancy in the contents of the Operations Manual, in that it repeats messages included in the Run Description. However, the potential benefits of such a list outweigh the inconvenience of maintaining duplicate information.

3.5 Non-Routine Procedures

Section 4 of the content guidelines should be expanded to emphasize the control responsibilities of operations personnel. The following outline suggests one form such an expansion might take:

4. NON-ROUTINE PROCEDURES

   4.1. Emergencies and System Failures.
   Identify types or causes of potential system failures and emergencies. Define for each the responsibilities of operations personnel. Include such information as the features and operation of safety/security devices; emergency priorities; and problem reporting procedures.

   4.2. Switchover to a Back-up System.
   Describe any available back-up systems and the conditions and procedures for their use.

   4.3. Turnover to Support Groups.
   Describe the conditions and procedures for transferring responsibility for operation of the system to appropriate support groups, such as maintenance programmers, computer engineers, network control analysts, testing and installation staff, etc.

Although some of this information might be more pertinent in an administrative manual, including it in the Operations Manual would ensure its availability to computer operators. Content guidelines reflecting the need for safety, security, and contingency procedures should minimize the occurrence of disasters and the time required to recover from them.

3.6 Benefits of Revised Content Guidelines

The four revisions to the Operations Manual, outlined here, stress the need for increased understanding and awareness on the part of operations personnel in an online system environment. As Guldentops has observed, "Operating online systems often becomes a tedious and boring job needing few interventions and little initiative, but requiring constant alertness."[3] Although actual operating tasks become simpler, the required knowledge and skill level necessary to ensure high system performance become more complex.

Attention to the needs of the operations audience -- both batch and online -- should result in better system performance and fewer documentation revisions.

4. REFERENCES

1. Schaeffer, Howard, Data Center Operations (Englewood Cliffs, N.J.: Prentice-Hall, 1981), p. 284.

2. Catania, Salvatoria C., "Designing an Efficient On-Line System," Auerbach Information Management Series (Pennsauken, N.J.: Auerbach Publishers, 1977), Portfolio No. 3-10-04, p. 2.

3. Guldentops, E., "Computer Audit and Control," Infotech State of the Art Report: Computer Audit and Control , Series 8, No. 8, (1980), p. 100.

Documentation for Operation Phase of Systems Life Cycle

Robert A. Larson

USDA, Forest Service

This paper is on software documentation requirements during the Operation Phase of automated systems life cycle in the Forest Service. Implementation of these requirements was started in 1978. This encompassess some five thousand programs in a distributed/dispersed environment. The Forest Service has a systems management process covering the systems life cycle which includes this phase. This process has been implemented in our national and field offices with significant success.

The documents to be presented come from the Forest Service Systems Management Manual and Automated Systems Management and Documentation Handbooks. This paper covers the document contents, associated experiences and future plans for managing software in the Operation Phase.

1. INTRODUCTION

The Forest Service, U. S. Department of Agriculture, has primary responsibility for protection and management of this nation's forest resources. There are three operating divisions: one is charged with the management of approximately 187 million acres of federally-owned National Forests, a second promotes forestry on state and private lands, and a third is a research arm. The Forest Service is comprised of 9 National Forest Regions, 2 State and Private Forestry Areas, 8 Research Stations and a Forest Products Laboratory. Each Region has from 13 to 20 National Forests and each Forest has from 4 to 6 Ranger Districts. In total there are 123 National Forest offices and 653 Ranger District offices within the Forest Service.

Forest Service management is delegated to the lowest appropriate level. It is essential that computing resources be provided to all organizational levels and that these resources be prudently managed. The central host computer used by the Forest Service is a UNIVAC 1100/84 located in Fort Collins, Colorado. The Forest Service also has computing and word processing facilities, including stand-alone facilities, located in the National Headquarters, Region, Area and Station offices, most Forests, and many Ranger Districts. There is a wide variety of outside computer usage through contracts and cooperative agreements with major universities. Most of the in-house facilities communicate with or through the host computer and range from non-intelligent terminals to full size computers. These facilities provide for the development and use of over 5000 programs in a distributed/dispersed environment.

In 1976 the Forest Service created a new Systems Mangement organization as the result of a study having input from management, users, and computer specialists. Three areas of need identified and emphasized in that study were for systems standards, documentation requirements and improved systems management. The Forest Service successfully implemented a systems management process covering the total systems life cycle in 1977.

Some standards and guidelines had been drafted by both in-house resources and consultants, but had not been implemented prior to January 1977. This material along with FIPS PUB 38 and DOD's Documentation Standards were used in developing the Forest Service documentation handbook. The work was accomplished through workshops and the use of personnel detailed from field units on specific project assignments to assure field input. Management and technical staffs at all organizational levels reviewed the completed work and it was published in November, 1978.

This paper presents Forest Service documentation requirements for the Operation Phase of the Systems Life Cycle. Several existing systems have been brought into the management process and into compliance with standards using these requirements.

Documentation is a key software mangement tool where both management and data processing facilities are widely dispersed. Our management process uses this documentation to support management and technical reviews throughout the systems life cycle (Figure 1).

2. RECOMMENDATION

There is a need for consistent Government-wide guidelines and standards covering documentation requirements in all phases of the systems life cycle. Currently only the Initiation and Development Phases are covered by FIPS PUB's. There are strong indications that FIPS PUB 38 needs to be updated. The Forest Service is in the process of updating their documentation requirements for all phases including the Operation Phase.

The purpose here is to present Operation Phase documentation, how it is being used and its effectiveness. The recommendation is that similar requirements or quidelines could and should be applied Government-wide at an early date. The Forest Service documents, after inclusion of proposed improvements, could be used as a starting point for development of a FIPS PUB Guideline for Operation Phase Documentation.

3. DISCUSSION

The Operation Phase starts immediately following formal system acceptance by users and management. The first step is full implementation following an approved Implementation Plan. It is preferred that a system be certified prior to this, but this has not always been possible. Completion of the Operation Phase occurs with a management decision to remove, replace or extensively modify the system. In the event of replacement or extensive modification the system should cycle back to the Initiation or Development Phase.

The documentation requirements have not seen as much use as anticipated, but what the Forest Service has seen is quite promising. Some of the improvements needed will be covered later in this discusssion. The primary software management documents involved with this phase are as follows:

1. Automated Systems Certification
2. Problem Report/Modification Request
3. Periodic Review and Evaluation
4. In-depth Review Plan
5. In-depth Review and Evaluation

These documentation requirements have assisted the Forest Service in cleaning up their libraries, bringing systems up to standard, reducing maintenance and improving the quality of existing systems.

The Project Acceptance document which is signed prior to implementation does not assure that the system is certifiable. In fact, in most instances there is still significant work to be done prior to certification. It does provide assurance that the system is acceptable and has the approval of management to be fully implemented. Management can then allow a reasonable time frame to meet the certification requirements while implementation is progress. There are few certified systems to date, but the Automated Systems Certifications (Figure 2) was not required for new systems until about a year ago. This document and other modifications were a result of attempting to assure compliance with Departmental and Forest Service requirements. There has been a fair start towards doing just that.

The Problem Report/Modification Request (Figure 3) with associated procedures and requirements were placed in effect about the same time as the certification document. This is a record of problems and modifications and the appropriate action taken. Prior to this there were complaints of problems reported and modifications requested for which there was no apparent response from the system support staff. Some were legitimate, some were not, but there was no documented evidence one way or the other. If this document has done nothing else, it has improved communications between the end users and the support staff.

Other maintenance procedures and documents may be used, but they must first be approved. Also, the alternative requirements and procedures must be included in the user and maintenance manuals. With this direction there has been noted improvement of systems maintenance records including the documentation of changes. Much more needs to be done but it is a good start. More people are now getting involved and are looking for further improvements.

One problem in the past was obsolete, undocumented and low usage programs in our libraries. To deal with this a requirement was established for periodic (minimum of once per year) review of all systems. With this a Periodic Review and Evaluation document (Figure 4) was developed and mandatory review criteria identified. This document was only partially successful, but it served to point out things to look for in existing systems.

The Forest Service is now looking at modifying the periodic review requirements and associated documents. This will involve partially automating the process by providing an annual report showing computer resource utilization, summary of problems and modification requests, and support staff comments for each system. This review will not require interviews or significant user involvement. Most of the burden of preparing this report falls on the systems support staff.

Based on a current need or the results of a periodic review, management may request an in-depth review. This review covers many of the same criteria as acceptance testing with heavy involvement of the developer, user, and management personnel. This review also looks at current needs and the system environment. An in-depth review may replace a periodic review.

The extensiveness of an in-depth review and the significance of resources utilized for that review requires an approved In-depth Review Plan (Figure 5) prior to the review. Upon completion of the review, an In-Depth Review and Evaluation Report (Figure 6) is presented to management. Management's decision at this point will be to (1) remove the system, (2) approve continuance, (3) approve modification, or (4) approve replacement.

This review has been effective in bringing existing systems into the management process. It has also created a management awareness that was not there before.

The Forest Service is currently looking into simplifying the process for review of smaller systems. To date, primary usage has been on large systems with identified problems or potential modification requirements.

This completes coverage of all of the required operation phase documents except for the review criteria. The review criteria are key to the pilot test, periodic and in-depth reviews and to system certification. Some users would like simple criteria like "does it run", but there is more to it than that. Major review criteria categories are identified in Figure 7. Figure 8 contains an example of detailed criteria for one category. Other systems dependent or management specific criteria may be added to this list for any of the reviews. Certification or recertification of a system requires full compliance with the mandatory pilot test criteria.

At the time these requirements were established there was very little reference material available or experience to draw from. FIPS PUB standards or guidelines would have been used if they had been available. The wheel should not have to be re-invented. Any NBS efforts to publish standards, guidelines, procedures, covering this segment of the data processing area should be supported. The sooner they can do this the better, easier and more cost effective it will be for all concerned. There should also be more flexibility and a faster release of new material and updates. Several standards and guidelines in the past have been obsolete before they were published.

4. CONCLUSION

Forest Service Operations Phase documentation requirements are far from perfect. However, they provide a good basis from which to start. With their past experience and that of others, the Forest Service hopes to significantly improve their requirements in this phase of the systems life cycle.

To effectively develop and implement data processing standards and guidelines it takes a concerted effort by management, developers and users. Without management backing many individuals will not read or use the material although it is to their benefit and that of the organization. Even with management backing the implementation process can be slow and sometimes painful.

As stated at a recent standards workshop, the six E's (Establish, Educate, Encourage, Enforce, Evaluate, and Enhance) are required for successful implementation and use of standards and guidelines. The weakest areas are Education and Encouragement. This is often due to insufficient resources such as people and money to do it effectively. Everyone is familiar with Establish and Enforce. In fact, so familiar that the need for emphasizing Evaluation and Enhancement is overlooked . The Forest Service anticipates Enhancement to the periodic reporting process and documentation in the near future. With that to be more effective, they are planning to improve on the Education and Encouragment of their staffs and management.

AUTOMATED SYSTEMS LIFE CYCLE								
PHASE		INITIATION		DEVELOPMENT				OPERATION
STAGE		PROPOSAL	FEASIBILITY	DEFINITION	DESIGN	PROGRAMMING	TEST	
Documentation Required See FSH 6609.54		1. Project Proposal (PP)	1. Project Proposal (Original, updated or modified) 2. Feasibility Study Report (FSR)	1. Automated Systems Inventory (ASI) 2. Functional and Data Requirements (FDR) 3. Development Plan (DP)	1. System/Subsystem Specifications (SS) 2. Program Specifications (PS) 3. Data Base Specifications (DBS) 4. Development Test Plan (DTP)	1. Test Results and Certification (TRC) 2. User's Manual (UM) 3. Operations Manual (OM) 4. Maintenance Manual (MM) 5. Pilot Test Plan (PTP)	1. Pilot Test Evaluation Report (PTE) 2. Implementation and Support Plan (ISP) 3. Project Acceptance Report (PA)	1. Maintenance documents (normally in the form of updates to other documents in System Documentation File) 2. Periodic Review and Evaluation Report (PRE) 3. In-depth Review Plan (IRP) 4. In-depth Review and Evaluation Report (IRE)
Staff Director and Program Staff Responsibilities	Primary Activities/Processes	1. Establish need or opportunity for an automated system. 2. Prepare and submit Project Proposal to Staff Director. 3. Staff Director reviews the Project Proposal and submits it with recommendations to management for approval.	1. Establish a System Documentation File. 2. Complete a Feasibility Study and Report. 3. Staff Director reviews the Feasibility Study Report and submits it with recommendations to management for approval.	1. Prepare and submit Automated Systems Inventory data. 2. Develop, analyze and verify detailed functional and data requirements. 3. Prepare Functional and Data Requirements document. 4. Update the Feasibility Study Report. 5. Establish an overall plan for development including documentation requirements. 6. Prepare the Development Plan. 7. Staff Director reviews the required documents and submits them with recommendations to management for approval.	1. Analyze the specific requirements and identify possible new requirements. 2. Establish alternative system designs and identify trade-offs. 3. Select best overall design and establish the details for that design, including non-computer procedures. 4. Prepare System/Subsystem Specifications. 5. Staff Director reviews and approves selected design. 6. Define (in detail) the programs, physical files, data bases and/or data base files, and any new data dictionaries/directories. 7. Prepare Program Specifications and/or Data Base Specifications. 8. Develop and prepare a Development Test Plan. 9. Development and user staffs inspect system design and plan for testing. 10. Update Development Plan. 11. Staff Director reviews the required documents and submits them with recommendations to management for approval.	1. Modularly code, test, debug and evaluate the programs, data bases and/or data base files. 2. Prepare Test Results and Certification document with Staff Director certification that the computation and other system procedures are appropriate and correct. 3. Prepare User, Operation and Maintenance Manuals and a Pilot Test Plan for testing the system in a user environment. 4. Update the Automated Systems Inventory. 5. Update pertinent documents in System Documentation File. 6. Staff Director reviews required documents and submits them with recommendations to management for approval.	1. Evaluate operation and maintenance of system to ensure requirements have been met. 2. Test the system in a field environment and correct problems. 3. Evaluate the system documentation, training and support. 4. Make a complete evaluation of the technical, operational and managerial aspects of the system. 5. Prepare Pilot Test and Evaluation Report. 6. Establish and prepare an Implementation and Support Plan. 7. Staff Director reviews the required documents and submits them with recommendations to management for approval.	1. Implement and maintain the system. 2. Provide user training. 3. Evaluate and modify the system as needs are identified and approvals obtained. 4. Keep an up-to-date maintenance log. 5. Keep Automated Systems Inventory updated. 6. Prepare or update material to cover changes and reviews. 7. Provide Periodic Review and Evaluation Reports annually to Staff Director for approval. 8. Conduct an In-depth Review when Line Management determines the need. a. Prepare In-depth Review Plan (IRP) b. Staff Director reviews the IRP and submits it to management for approval. c. With approval, an In-depth Review, Evaluation and Report (IRE) is completed. d. Staff Director reviews the IRE and submits it to management for approval.
Management Review and Approval Responsibilities		SYSTEMS COORDINATOR	SYSTEMS COORDINATOR	SYSTEMS COORDINATOR	SYSTEMS COORDINATOR	SYSTEMS COORDINATOR	SYSTEMS COORDINATOR	SYSTEMS COORDINATOR
	Coordination and Recommendations	SYSTEMS COORDINATING COUNCIL	SYSTEMS COORDINATING COUNCIL	Technical review and approvals by CT, OM, and CSA are required at each of these milestones. SYSTEMS COORDINATING COUNCIL	SYSTEMS COORDINATING COUNCIL (OPTIONAL)	SYSTEMS COORDINATING COUNCIL	SCC Note: Staff Director(s), and SCC prepare PA Report.	SYSTEMS COORDINATING COUNCIL
	Systems Management Approval	ASSOCIATE DEPUTY CHIEF, ADMINISTRATION	ASSOCIATE DEPUTY CHIEF, ADMINISTRATION	ASSOCIATE DEPUTY CHIEF, ADMINISTRATION	ASSOCIATE DEPUTY CHIEF ADMINISTRATION	ASSOCIATE DEPUTY CHIEF, ADMINISTRATION	ASSOCIATE DEPUTY CHIEF, ADMINISTRATION	ASSOCIATE DEPUTH CHIEF, ADMINISTRATION
	Program Approval	DEPUTY or CHIEF and STAFF	DEPUTY or CHIEF and STAFF	DEPUTY or CHIEF and STAFF	DEPUTY or CHIEF and STAFF	DEPUTY or CHIEF and STAFF	DEPUTY or CHIEF and STAFF	DEPUTY or CHIEF and STAFF

Figure 1

AUTOMATED SYSTEM CERTIFICATIONS

SYSTEM NAME_____

SYSTEM-ID (3-Char.)_____

In addition to the approvals which have been given to the development stages for this system, the following specific certification are documented for the system record:

TECHNICAL ASSURANCES

The accuracy of modeling assumptions and the algorithms of the system are deemed to be appropriate. Assurances that all applicable national standards and security requirements have been met are hereby provided.

_____ _____

Director, Support Staff Date

SYSTEM CERTIFICATION

Certification of the adequacy of security requirements (such as: FSH 6609.33, OMB Cir. No. A-71), National Standards Compliance, Modeling Assumptions, algorithms, and documentation is hereby provided.

_____ _____

Associate Deputy Chief for Administration Date

Figure 2

U.S. Department of Agriculture
Forest Service

AUTOMATED SYSTEM PROBLEM REPORT/MODIFICATION REQUEST

(Instructions Reference FSM 6620)

PART I – REQUESTING UNIT

1. Originator's Name	2. Unit	3. Date

4. Address *(Include City, State, and Zip Code)*

5. System Name	6. System ID *(3 characters)*

7. Type of Request *(X appropriate box)*

☐ Problem or Error ☐ Suggested Modification ☐ Information Only

8. Description

9. Materials Attached

PART II – SUPPORTING UNIT

1. Evaluator's Name	2. Date of Evaluation

3. Evaluation

4. Estimated Cost

5. Action Taken

6. Actual Cost

7. Action Taken By	8. Date Action Completed	9. Date Returned to Requesting Unit

FS-6600-2 (1/80)

Figure 3

PERIODIC REVIEW AND EVALUATION REPORT

Contents

Figure 4

IN-DEPTH REVIEW PLAN

Contents

Figure 5

IN-DEPTH REVIEW AND EVALUATION REPORT

Contents

Figure 6

MAJOR EVALUATION CRITERIA CATEGORIES

I. MANAGERIAL

A. Functions of the Model for Forest Service Management
B. Management by Objectives Operation
C. Process Interfaces
D. Decision Utility

II. OPERATIONAL

A. Documentation
B. Cost of installation and operation
C. Training
D. System Performance
E. Data Storage and Handling
F. Hardware Requirements

III. TECHNICAL

A. Analytical Aspects
B. Systems Design and Programming
C. Systems Maintenance

Figure 7

EVALUATION CRITERIA GUIDELINES	EVALUATION TYPE 1/		
	PILOT TEST	PERIODIC REVIEW	IN-DEPTH REVIEW
II. OPERATIONAL			
D. System Performance			
1. Meets the specified needs of intended users at all levels.	M	M	M
2. Predetermined runstreams used are sufficient, minimizing need for new or special runstream generation by the user.	M	M	M
3. Users at all levels support the systems approach and are satisfied with the performance of the system.	M	M	M
4. All functions of the system are performed in a cost effective manner.	M	M	M
5. Required input forms are easily understood by users and can be completed with minimum effort.	M	M	M
6. Required input forms minimize data entry efforts.	M	M	M
7. Systems generated reports are readable and easily understood by users.	M	M	M
8. Include other performance functions identified for the system being evaluated.	IA	IA	IA

1/ NA = Not applicable IA = If applicable M = Mandatory

Figure 8

A Proposed Guideline for Documentation of Computer Programs and Automated Data Systems for the Operations Phase

Thomas M. Kurihara, CDP

U.S. Department of Transportation
Office of Information Systems and Telecommunications Policy

This paper presents a proposal for the third publication of a set of Federal Information Processing Standards (FIPS) publications describing the documentation content guidelines for computer programs and automated data systems for the operations phase. The proposed guideline is necessary to complete the work begun by FIPS Task Group 14 and is in response to the General Accounting Office Report to the Congress of October 8, 1974, "Improvement Needed in Documenting Computer Systems."

Keywords: Documentation, Federal Information Processing Standards (FIPS), Operations Phase, Computer Programs, and Automated Data Systems

1. INTRODUCTION

1.1 Proposal

This paper presents a proposal to prepare and publish a guideline for documentation of computer programs and automated data systems for the operations phase. The purpose of the guideline is to describe the content guidelines for each document type needed during the operations phase of a software life cycle. Federal Information Processing Standards(FIPS) Publication 38, "Guidelines for Documentation of Computer Programs and Automated Data Systems,"(1) covers the document types prepared during the development phase of a software life cycle; and FIPS Publication 64, "Guidelines for Documentation of Computer Programs and Automated Data Systems for the Initiation Phase,"(2) covers the document types prepared during the initiation phase of a software life cycle.

1.2 Need

The General Accounting Office (GAO) in 1971(3) and in 1974(4) cited the need for improved documentation of computer systems. In particular, GAO cited increased costs of operations and weakened management controls resulting from inadequate documentation of computer systems. The author believes that the publication of documentation content guidelines for the operations phase will assist managers,users, auditors, and software support staff to establish and document post-implementation review functions and control changes to operational computer systems.

1.3 Interest

In 1977, the FIPS task Group 14 Chairperson requested an affirmation of interest in further task force work and recommendations for specific document types and content guidelines from task group members. The majority of respondents cited an interest and need for a post-implementation review document, project development notebook, and standards and procedures manual. Of the document types most frequently mentioned, only the post-

implementation review document is included in this proposal.

1.4 FIPS Task Group 14 Plans

The Chairperson of FIPS Task Group 14 included in the scope and program of work for 1978 the development of documentation content guidelines for the operations phase. Because of th time required to complete the work on the draft FIPS Publication 64 and the subsequent disbanding of FIPS task groups, no work was started. The scope and program of work recommended were:

Scope: To develop additional documentation content guidelines for documentation of computer software during the operations phase. Include for consideration the documents resulting from the user acceptance of computer programs and automated data systems, requests for changes to operational systems and related data bases, post-implementation reviews and audits, performance measurement and evaluation, and conversion studies for data, computer programs and automated data systems.

Program of Work:

- survey current policies and practices of Federal executive agencies for documentation in the operations phase, including the use of automated documentation aids and productivity improvement tools
- assemble representative examples of documentation content guidelines, forms, and procedures that can be used as a starting point for the task group work
- develop a FIPS publication promulgating content guidelines similar to and supplementing FIPS Publications 38 and 64; consisting of

  Part I - Documentation within the Software Life Cycle

  Part II - Documentation Considerations

  Part III - Content Guidelines for Document Types.

1.5 Standards Workshop Recommendations

In the summer of 1980, members of the Washington, D.C. metropolitan area Federal ADP Users Group (FADPUG) Special Interest Group on ADP Standards and Quality Assurance (SIGSTD/QA) organized a workshop to review FIPS Publication 38. The result of the workshop sessions was "A Proposed Documentation Standard Based on a System Decomposition and Information Base Approach," authored by Mr. Saul Zaveler.(5) The workshop details were reported in a paper presented at the Software Engineering Standards Applications Workshop, August 18-20, 1981, in San Francisco, "Observations on Documentation Standards Revision: FIPS Pub 38 After Four Years."(6) The proposed guidelines preparation should include consideration of the recommendations and concerns developed during the SIGSTD/QA workshop on the approach taken to prepare FIPS Publication 38 and its contents and structure.

2. RECOMMENDATIONS

2.1 Document Content Guidelines

The document types included in the guidelines should be a documentation record of management, development staff, and user activities during the operations phase, and should include the post-implementation review document, change control document, and performance evaluation document.

2.2 Documentation Standards Manual

The documentation content guidelines issued by the National Bureau of Standards (NBS) as FIPS and those issued as part of Reports on Computer Science and Technology, NBS Special publication 500-nn series such as the "Computer Model Documentation Guide," NBS Special Publication 500-73,(7) should be consolidated into a software engineering documentation standards manual.

3. Guidelines for Documentation of Computer Programs and Automated Data Systems for the Operations Phase

3.1 Scope

Computer programs and automated data systems evolve in phases, from when an idea to create the software emerges through the time that software produces the required results. Many different terms are currently used to identify these phases and the stages within the phases. Three phases were described in FIPS Publication 38--initiation, development, and operation. FIPS Publication 64 addresses content guidelines for the initiation phase; FIPS Publication 38 addresses content guidelines for the development phase. The proposed FIPS publication addresses the operations phase.

3.2 Stages and Key Activities

The operations phase can be divided into stages, for example:

OPERATIONS PHASE				
ACCEPTANCE STAGE	OPERATION STAGE	MODIFICATION STAGE	RECERTIFICATION STAGE	TERMINATION STAGE
user acceptance report*		user acceptance report*		
change control document ———				
	performance measurement and evaluation document ———			
post-implementation review document ———				
strategic ADP plan ———				

*part of the post-implementation review document

3.3 Proposed Guideline Contents

The proposed content guidelines are not intended to be a complete set of document types which should be determined by the consensus of the working group. The proposed content structure is consistent with that of FIPS Publications 38 and 64.

Part I - Documentation within the Software Life Cycle

Part II - Documentation Considerations

Part III - Activities of the Operations Phase

Part IV - Document Type Content

A. Post-implementation Review Document

B. Performance Measurement and Evaluation Document

C. Change Control Document

D. Strategic ADP Plan

Part V - Glossary

Part VI - Documentation and Review Methodologies and Aids

4. POST-IMPLEMENTATION REVIEW DOCUMENT

4.1 Purposes

The purposes of the post-implementation review are to examine a delivered and accepted software system and determine adequacy of the technical design, development process, operational performance, cost, and user satisfaction; to assess periodically, e.g., annually, its overall usefulness; to plan major modifications; and to plan for system termination.

4.2 Scope

The scope of the review is an assessment of the successes and shortcomings of project performance and system performance. Users, software development staff, operations staff, and auditors are involved. Key considerations include user acceptance, system development processes, lessons learned, operating efficiency, achievement of expected benefits, and goal setting procedures for system expectations.

4.3 Objective

The objective of the post-implementation review is to determine whether:

- preliminary studies were complete and realistic
- implementation progressed according to plan
- original cost/benefit analysis projections were accurate and are as projected
- operations, documentation, and system products are adequate
- system performance and schedule expectations were realistic and are being realized

- changes and modifications were adequately handled and change control procedures have been implemented
- the system is of value in supporting organizational mission needs.

4.4 Content Guidelines

The proposed content guidelines include those subject areas that might be reviewed for adequacy.

Section 1. GENERAL INFORMATION
- Summary
- Environment
- References

Section 2. MANAGEMENT SUMMARY
- Requirements
- Objectives
- Assumptions and Constraints
- Methodology
- Evaluation Criteria
- Recommendations

Section 3. GENERAL EVALUATION
- User Acceptance
- Completeness of Preliminary Studies
- System Products and Data
- Systems Documentation and User Procedures
- Security and Accountability
- Performance measurement and Evaluation
- Operations and Facilities
- Lessons Learned

Section 4. FEASIBILITY STUDY AND IMPLEMENTATION
- Current Costs and Benefits
- Projected Costs and Benefits
- Explanation of Differences

Section 5. SYSTEM DEVELOPMENT PROCESS
- Description of System Management and Development Process
- Problems Encountered and Solutions
- Lessons Learned

Section 6. USER PARTICIPATION PROCESS
- Description of User Involvement in System Management and Development Process
- User Acceptance Procedures
- Problems Encountered and Solutions
- Lessons Learned

Section 7. STRATEGIC ADP PLANNING PROCESS
- Description of Planning Process for Near-Term and Long-Term
- Resource Requirements

Section 8. POST-IMPLEMENTATION REVIEW REPORT TO STEERING COMMITTEE

5. PERFORMANCE MEASUREMENT AND EVALUATION DOCUMENT

5.1 Purpose

The purpose of the performance measurement and evaluation is to compare software system performance measures and the results of performance measurement and evaluation activities with established criteria and expected results.

5.2 Scope

The scope of the document covers the results of planned and scheduled software system performance evaluations during the operations phase using established criteria and procedures. The results of the evaluation activities assist organizations in making decisions during the operations phase related to strategic planning, resource allocation, utilization, system modifications, operations, recertification, and termination.

5.3 Objective

The General Services Administration (GSA) in November 1978 published "Management Guidance for Developing and Installing an ADP Performance Management Program."(8) The objective of the GSA program is to improve the effectiveness of ADP systems. Because the program is primarily oriented toward hardware systems, additional management guidance is needed for software systems.

5.4 Content Guidelines

The proposed content guidelines include those major areas applicable for performance measurement and evaluation.

Section 1. GENERAL INFORMATION

Section 2. MANAGEMENT SUMMARY

Section 3. MEASUREMENT METHODOLOGY

Section 4. EVALUATION OF MEASUREMENT DATA

Section 5. REVISED EVALUATION CRITERIA AND MEASURES

Section 6. RECOMMENDATIONS

Appendices - Details of Performance Measurements and Analysis

6. CHANGE CONTROL DOCUMENT

6.1 Purpose

The purpose of the change control document is to establish an orderly, manageable process for identifying changes, classifying changes, evaluating the impact of changes, and approving the changes to modify operational software systems.

6.2 Scope

The scope of the document includes a description of organization, activities, and procedures for change control and the documentation required to manage and control changes to software systems effectively. Implementation of changes resulting from performance measurement and evaluation activities and conversion studies may be included under change control activities.

6.3 Content Guidelines

The proposed content guidelines include those major areas applicable to change control.

Section 1. GENERAL INFORMATION

Section 2. CHANGE CONTROL POLICIES

Section 3. APPLICABILITY AND IMPLEMENTATION POLICIES

Section 4. CHANGE IMPACT ANALYSIS METHOD

Section 5. CHANGE CONTROL DOCUMENTS
Change Identification
Change Control
Status Accounting

Section 6. GLOSSARY

Appendices - Change Control Organization, Procedures and Forms

6.4 Related Standardization Activities

The Electronics Industries Association and the Institute of Electrical and Electronic Engineers are involved in standardization acitivies for data management and configuration management in Department of Defense contracting activities.

7. STRATEGIC ADP PLAN SYSTEMS

7.1 Purpose

The purpose of the plan is to document the strategic automatic data processing planning activities which coordinate systems planning, systems development, systems acquisition, and system operations. The activities and resulting plan show what operational and planned software systems support mission needs and how available and expected resources are allocated in the time period covered by the plan.

7.2 Scope

The scope of the ADP plan includes the software and hardware systems needed to run the software, people, dollars, and facilities needed to plan, develop, acquire, manage, and use computer-based information systems. The plan should include information resulting from planning activities at each organizational level.

7.3 Content Guidelines

The proposed content guidelines include those major areas applicable to strategic planning.

Section 1. GENERAL INFORMATION

Section 2. MANAGEMENT SUMMARY

Section 3. PLANNING POLICIES

Section 4. PLANNING METHODOLOGY

Section 5. ORGANIZATION MISSION NEEDS

Section 6. TECHNOLOGY ASSESSMENT

Section 7. CONCEPT OF OPERATIONS
Development
Acquisition
Systems Operation and Management
User Involvement

Section 8. ALLOCATION OF RESOURCES

Appendices - Details of Studies, Analysis, and Resource Allocation

8. CONCLUSION

Well written documentation does more than describe the programming language code. It communicates information about:

- intended purpose and use of software systems
- transformation of requirements into a design for development of a software system
- development process
- performance criteria and expected changes
- controls for data and for operation of the software system
- limitations, constraints and assumptions
- user acceptance criteria and acceptance procedures
- operation for auditing and verification of software systems.

FIPS Publications 64 and 38 describe the document content guidelines for the initiation and development phases. Another FIPS publication is needed to communicate information about the activities related to the operations phase.

A final note: Documentation standards and guidelines should be developed after the management, technical, and user activities are described. Once these activities are described and are accepted as good, uniform practices to be followed for developing reliable, high-quality software, the documentation content guidelines will become more useful as an aid for effective communications. With the FIPS for the management, technical and user practices, documentation standards and guidelines will be more valuable to the Federal community. Since there appears to be very little emphasis on defining uniform practices in software development, the development of software documentation content guidelines is needed to encourage better communication about software systems during the operations phase.

9. REFERENCES

1 Federal Information Processing Standards Publication 38, National Bureau of Standards, Washington, D.C., February 15, 1976, 50 pages.

2 Federal Information Processing Standards Publication 64, National Bureau of Standards, Washington, D.C., August 1, 1979, 54 pages.

3 The Comptroller General of the United States, Report to The Congress, "Case Studies of Auditing in a Computer-based Systems Environment," Washington, D.C., June 1971.

4 The Comptroller General of the United States, Report to The Congress, "Improvement Needed in Documenting Computer Systems," Washington, D.C., B-115369, October 8, 1974, 46 pages.

5 Zaveler, Saul, "A Proposed Documentation Standard Based on a System Decomposition and Information Base Approach," National Bureau of Standards Software Documentation Workshop, March 3, 1982, Gaithersburg, Maryland, March 1982, 6 pages.

6 Kurihara, T., Redwine, S.T., Jr., and Zaveler, S., "Observations on Documentation Standards Revision: FIPS Pub 38 After Four Years," Institute of Electrical and Electronic Engineers, Proceedings--Software Engineering Standards Application Workshop, August 18-20, 1981, San Francisco.

7 National Bureau of Standards Special Publication 500-73, "Computer Model Documentation Guide," Reports on Computer Science and Technology, National Bureau of Standards, Washington, D.C., January 1981, 56 pages.

8 General Services Administration, "Management Guidance for Developing and Installing an ADP Performance Management Program," General Services Administration, Automated Data and Telecommunications Services, Washington, D.C., November 1978.

Session B: Documentation for Operation and Maintenance

Synopsis of Discussion

Nancy Mae Bonney

Dynamac Corporation

The following comments were made by audience participants:

We only need general guidelines, like a checklist of items that system developers should not overlook. Because of the rapidly changing ADP environment (e.g., personal computers, user-oriented languages, etc.), specific standards for on-line user-driven systems might be obsolete before they were published. Perhaps the situation has to 'crystallize' for a while longer. -- Carol Uri, Federal Communications Commission.

Most of the comments and presentations were geared to business systems or on-line business applications. There is a great need for software documentation and maintenance procedures for on-line machine control, distributed process control and interactive systems. -- John Maupe, U.S. Postal Service.

Research is needed in presentation syle and in writing style. The results should be applied as an addendum to each documentation content guideline issued in the FIPS program. Writing style and presentation must consider management needs, audience, document type, and application. We need more than the GPO Style Manual. Tom Kurihara, U.S. Department of Transportation

I still fail to comprehend how a single universal standard can be all things to all people. It would seem that documentation content guidelines must be developed in-house for a specific group of users. Standards are of course necessary, but must be developed toward unique environments. -- Anonymous

FIPS PUB 38 guidelines for the Users Manual, Operation Manual, and Program Maintenance Manual should provide a recommended documentation structure and content that allow for the manuals to be easily maintained over the system's lifetime. Currently, amendments to the users manual turn out to be quite large because of the documentation structure. Also, the user community now needs different types of manuals: a management type, a terminal instructions type, and in some cases, a distributed processing type for field users. -- Carolyn Cowden, Calculon Corporation

Documentation should be targeted to 'user levels' in the next publication of the FIPS documentation standards. -- Charles R. Cooke, U.S. Department of Health and Human Services

We need to distinguish between system management (operation phase) and project management (development phase). These are quite different. We need to maintain the documents produced in the development phase in the operation phase (e.g., the Operations Users Manual, and Maintenance Manual). We need to have in-code (in program) documentation developed during the development phase and updated during the operation phase. Program technical documentation is more beneficial (maintainable) when in actual program code. -- Michael A. Regardie, U.S. Navy Recuriting Command

We should follow all the recommendations presented, and be aware of:

1) On-line vs. batch (environments).
2) Computer operator vs. the terminal operator as users.
3) User operation must be broken down also into; a) clerical, b) data entry, c) management, d) other. -- F. Colison, Defense Logistics Agency

I definitely recommend a separate guideline for the operation phase. Such a document should address the areas of:

a) Change/modification process (software).
b) Ongoing life of User and Operations guides.
c) Multiple user audiences.

Basically, I recommend something similar to the content guidelines proposed by T. M. Kurihara. -- Rod Smart, U.S. Department of the Interior

I agree with Mr. Kurihara's recommendations to address different modes of operation and different audiences but in a combined publication of FIPS PUBS 30, 38, and 64. The addition of hardware configuration and error message documentation in the revision would be very helpful also. -- Bonita Condon, National Institutes of Health

FIPS PUB 38 should be expanded to cover the operation phase, but I would stress the idea of using it as a guide. The audience must be the primary consideration. -- Cathy Mason, Calculon Corporation

Other points were made during the session discussions, including:

- Will management support quality documentation considering its potential cost?

- Following FIPS PUB 38 as a rule will not result in quality documentation.

- From the users' standpoint, the documentation is either usable or not; whether it meets FIPS guidelines is a different issue.

- Documentation as well as programs must be tested; this is one way to determine the quality of documentation.

- Will change control work in urgent situations? If standard change control is bypassed in urgent situations, how does one make sure the appropriate procedures are followed after the fact?

- Standards have frequently been developed so that they are easy to implement and maintain, rather than useful to the user audiences.

- Each government agency should develop their own standards based on FIPS documentation guidelines; telling a contractor to "use FIPS PUB 38" does not provide enough guidance.

- The FIPS documentation publications are guidelines only, and are not a substitute for good management decisions about system documentation.

The conclusions of the session indicated:

- the necessity for management and personal commitment for quality documentation,
- the FIPS PUBS are content guidelines, not standards,
- we must write to the audience,
- the coverage of the existing FIPS documentation guidelines needs to be broadened, and
- a revision and combination of FIPS PUBS 64 and 38, to include documentation guidelines for the operation phase, would be useful and appropriate.

SESSION C: Tools for Improved Documentation

Introduction

Moderator: Raymond C. Houghton, Jr.

Institute for Computer Sciences and Technology
National Bureau of Standards

In this session, there is a discussion of a good cross section of the types and applications of documentation tools. It includes tools that are currently available for application. It includes tools which address documentation issues for specific phases of the life cycle or for multi-phases of the life cycle. More important, the tools discussed cover the entire life cycle from requirements specification to code maintenance.

The first paper by S. Lee Henry provides a good introduction to the issues associated with the effects of technology on documentation and more important, on documentation standards. This discussion leads nicely into the specific tool coverage of the next four papers.

A. Malhotra (et. al.) brings back a familiar phrase that was once assigned to COBOL, that of "executable documentation". This paper, however, discusses a language that allows its user to work at a much higher level of abstraction. This higher level does bring us closer to the problem domain and consequently may actually bring us closer to "executable documentation".

T. C. Ting shows us that documentation can also be a subject of academic interest. He discusses a tool currently under development at the Worcester Polytechnic Institute that integrates documentation development into the earlier phases of software development.

Bruce Blum shows us how a system can maintain documentation throughout the life cycle. His system has the advantage of addressing a very specific application area, thus making it possible to automate much of the development and maintenance process.

Finally, Linda Lawrie discusses a "real world" tool that generates documentation from user-embedded comments and the code itself. An important issue that she brings out in her paper and that brings us back to the theme of the conference is that documentation standards can be cumbersome when they are applied to documentation systems. It is important for documentation standards to be flexible so that they do not stifle technological progress.

State-of-the-Art Documentation
What is it?
How does it Affect Documentation Standards?

S. Lee Henry

American Management Systems, Inc.

"State-of-the-art", when applied to documentation, describes a process as well as a product. State-of-the-art documentation is accurate, usable and easy to update. It conserves effort in the design, programming and maintenance of computer systems. This paper discusses the effects of software technology on current documentation format and content standards and suggests procedural guidelines which aid in the preparation and maintenance of state-of-the-art documentation.

Keywords: Procedures; Guidelines; Software; Compatibility

1. INTRODUCTION

Software documentation is written to protect the investment made by system designers and programmers. Anyone who weighs the cost of documentation against the savings it produces in system modification and user training, recognizes that documentation is an integral part of software development. In addition, state-of-the-art documentation makes it possible to better centralize development efforts, reduce programmers' workload, facilitate testing and maintain a clearing house of current information during system development and maintenance. The state-of-the-art of documentation has, therefore, come to have several important requisites.

One is that it must be cost-effective to produce. The cost of documentation should be proportional to the cost of programming. It is not unusual for the amount and type of documentation to be included in software contracts. This places a very real limit on what can be spent producing it.

Another is that it must be accurate. Good documentation can give a good system the competitive edge it needs in the software marketplace. In addition, with the rapid turnover in data processing personnel, documentation often is relied upon to transmit information from one employee "generation" to the next.

Lastly, and never leastly, it must be usable. Information must be in a format such that it is easy to find and readable. Principles of human engineering applied to documentation suggest that document formats be standardized, levels of detail be closely matched to the need of the intended audience and figures and charts be used profusely.

2. CURRENT TOOLS

Current software tools make it possible to adhere to format standards and increase the cost-effectiveness, accuracy, and usability of documents. The technology itself offers a great deal of compatibility with current FIPS guidelines for document format and content. Although differences between the tools may dictate some elements of format, each offers extremely important advantages.

Word processors allow straight-forward control over the format of documents through the "what you see is what you get" approach. Setting up page formats, tabs, margins, etc. is visually re-inforced as the text image is transformed on the screen.

Text editors, on the other hand, take the "set it up and run it through" approach. The formatting commands are imbedded throughout the text, and the resultant text is available after processing. Most text processors automatically number sections as well. Some can create tables of contents and indexes.

What all text and word processors have in common, of course, is the obvious advantage of being able to modify a document without the need for messy or tedious manual correction.

It is sometimes possible to interface these tools with sophisticated output devices -- such as laser printers and phototypesetting equipment -- enhancing the attractiveness of documents at reasonable cost.

Automatic documentation programs, on the other hand, are an entirely different kind of tool. They use the software itself as input and vary greatly in the kind and amount of data they produce. Some supply information on the files used by the software while others draw flowcharts. By extracting information from the program being documented, they help to ensure the utility of the resultant documentation.

The use of word processors, text processors, automatic documentation tools and computer-controlled output devices requires technical competence with text processors and file handling utilities as well. In order to reduce the complexities and cost of developing standard documentation and extracting information from the software, standard procedures should be developed that are commensurate with standards applied to the development of software.

3. SUGGESTED PROCEDURAL STANDARDS

3.1 Notation Conventions

Notation conventions add clarity to software documentation. Whether one is marking the difference between user-entered and system-generated data on a terminal, or noting required and optional parts of syntax, notation conventions convey a lot of information to a user in a brief format.

Current tools provide several means with which notation conventions can be expressed. Upper and lower case conventions can be maintained, bold-face and regular type can often be used, or the absence and presence of underlining can make the distinction.

Whatever convention is decided upon should be used consistently and should be one that can be expressed within the document file itself, rather than one which requires manual "fixes" later. The advantage that text and word processors have over manual methods of documentation production is that commands which "carry" the information concerning the upper/lowercase, type-face or underlining convention can be part of the document file. This makes it possible to modify the documentation without having to rethink the examples or syntax descriptions.

3.2 Inclusion of Working Examples

Another of the benefits of software-based documentation is that it permits the user to imbed tested examples (commands, inquiries, command files, JCL, etc.) into documentation without having to enter them manually. This saves a lot of effort, especially if it only takes a single command to pull the example into the text of the document. It also guarantees the accuracy of the document. Little is more discouraging to a new user than to try an example from the document and find out it doesn't work.

3.3 Standard Sections

Just as examples can be inserted into a document file, standard sections, such as notation conventions, terminology, or system overviews, can be prepared separately and imbedded in any document which requires them.

Standard sections strengthen the apparent relationship between documents in a "package" and make each of the documents more self-contained.

3.4 Software and Documentation Update Procedures

Updates to existing documents should also follow standardized procedures. One of the first and most important of these is a well-publicized cycle for documentation updates, coordinated with software releases or some annual period.

Standard procedures for filing and fixing "bugs" in both the software and the documentation can be used to simplify and control the process of modification. Problems should be submitted on standard forms which make it easy for a user to clearly identify the problem he is having. He should know what kind of data to submit with the form, know where to send it, and expect to get feed-back from someone responsible for program maintenance. One procedure for using current tools for this update cycle is to set up a special data base for "action items". This allows the problem spots to be randomly surveyed. Specific problems an be routed to those responsible for fixing the software or modifying the documentation. Reports can be routinely distributed to inform everyone involved of the current status of the software. Further, action items can be prioritized to add managament control over the modification of both software and documentation. When problems are fixed, a copy of the original form can be returned to the person who sent it in. This encourages those who give the system its most thorough test to keep the development and maintenance groups informed of its performance.

3.5 Back-up and Archival Storage of Documents

The use of text processors also requires development of standard procedures for back-up and archival of document files. A system crash or user error cannot be allowed to destroy weeks of work. Documents can be backed up to tape, disk or diskette depending on the system being used. Documents can then be restored from backup copies if current files are damaged.

OCR scanning further permits documents which are printed in OCR type-face to be re-entered from hard-copy.

3.6 Documentation Specialists and the Software Team

State-of-the-art documentation involves a coordination of programmers and writers in a "team" effort. Techniques used in the development, testing, and maintenance of software often parallel those used in development, testing, and maintenance of documentation. The documentation specialist should, therefore, be a member of the software development team, familiar with the tools at his disposal.

In addition, the more familiar he is with the software, the more likely he will be able to serve as a source of current information during planning and development. Documentation that is produced during development can serve as an important means of information transfer and a way of reliably capturing decisions which are made as well as those that still need to be made.

3.7 Internal Documentation During System Development

Programmers, at the same time, should always be required to document their work. Flowcharts, logic diagrams and tables should be produced whenever possible. These shortened versions of system information can be re-used as figures or appendices in the final documentation when appropriate.

Throughout system development, the growing body of documentation should be available to documentation specialists, programmers and management. It can be used to monitor progress and review development goals.

3.8 Testing the Documentation

Another important advantage of the sophisticated documentation specialist and his role in the software development team is that he can "test" the documentation by applying what is written. This, of course, provides a further test of the system too. It helps to ensure the integrity of the document and guarantees that the writer has the proper "slant" on the system.

4. CONCLUSION

The software-based production of documentation can greatly improve its accuracy, maintainability, and, often, its attractiveness -- so much so that it is becoming more the rule than the exception in the software industry. Production of state-of-the-art documentation, however, requires additional control over costs and production procedures which parallel that applied to the development of software.

The EAS-E Approach to Documentation.

A. Malhotra, H.M. Markowitz and D.P. Pazel

IBM Thomas J. Watson Research Center,
P.O. Box 218, Yorktown Heights, N.Y. 10598

ABSTRACT: EAS-E is a programming language integrated with a database management system that is under development at the IBM Thomas J. Watson Research Center. This paper discusses the EAS-E approach to program documentation. EAS-E programs consist of high-level operations on entities, attributes and sets. The syntax has been designed to be compact and readable. This paper compares EAS-E programs to programs in PL/I-DL/I and PL/I-SQL and shows that EAS-E programs are shorter and have much less non-problem-related code. Thus, they can be viewed as "executable documentation".

Keywords: English-Like, Self Documenting, Programming Language

1. INTRODUCTION

EAS-E (pronounced EASY) is an interactive programming language integrated with a database management system that is under development for VM/370 at the IBM Thomas J. Watson Research Center. EAS-E provides a full-screen input/output facility through an interface with DMS/CMS (6), a non-procedural facility for "browsing" the database (12), and a report generator. The EAS-E database management system (9) is capable of efficiently supporting very large databases stored on multiple DASD extents. It supports locking, detects deadlocks and protects the integrity of the database against hardware and software crashes. In this paper we shall only concern ourselves with the features that make EAS-E programs compact and readable. These are discussed below:

A. *The Entity, Attribute and Set (EAS) Model.* The application or system to be implemented is analyzed in terms of the Entities, Attributes and Sets that determine its status at a given point in time. In section 3.2 we show that a wide variety of data structures are either special kinds of sets or are almost immediately expressible as EAS structures. The simplicity of the basic concepts makes it possible to document the EAS status of a system or application in a simple manner.

Programs generate and maintain this status. They consist of higher-level operations on entities, attributes and sets. Since the statements of the program relate directly to the concepts used in analyzing the application or system, their intentions are usually quite clear.

B. *Powerful Integrated Commands.* The program statements operate on entities, attributes and sets. For example, they may create an entity, assign values to its attributes, file it into a set, and later find it on the basis of these attributes. These high level commands are translated by the EAS-E software into lower level operations that communicate with the database manager, search and maintain set structures, etc. Thus, the programmer works in terms of higher level concepts and need not be concerned with low level details like the management of pointers in data structures. EAS-E is an integrated language in that the programmer refers to entities in main storage and in the database in exactly the same manner. This simplifies programming and also makes programs more compact.

C. *English-Like Syntax.* The commands that specify the operations on entities, attributes and sets are written as English-like sentences. This makes them easy to read and understand by non-programmers as well as programmers. We shall show that, despite EAS-E's English-like syntax, EAS-E programs are more compact than equivalent programs in other languages.

It has become clear that program documentation that is produced separate from and in addition to the executable code does not work. It does not get updated when the code is updated, particularly under time pressure, and is usually out

of date and misleading. We believe that the compact, English-like EAS-E programs are self-documenting and obviate the need for program documentation.

Section 2 discusses the EAS model and a simple method of documenting EAS structures. Section 3 discusses some features of the EAS-E language. Section 4, on Integrated Language, compares EAS-E with two leading database languages. In these comparisons we are primarily interested in the size and the understandability of the source code. We shall show that the EAS-E programs are compact because several functions are taken care of by system software instead of being coded by the programmer.

2. THE EAS MODEL

Webster's Unabridged Dictionary (second edition) defines an entity as "that which has reality and distinctness of being either in fact or for thought ..." In an EAS-E representation, it is usually some "thing" (account, check, job) of the real world to be represented in the database. The attributes of an entity can be considered to be its properties or characteristics. At any instant in time an attribute has at most one value or it may be undefined.

In EAS-E, as in SIMSCRIPT (7) and DBTG (3) , a set is an ordered collection of zero, one, or more entities which is owned by some entity. To illustrate: each account owns a set of current transactions, i.e., checks and deposits which have occurred since the last monthly statement. Thus, the account for John Smith owns one such set, that for Mary Jones owns another such set, etc. In general, we say that each account owns its current transaction set. Accounts may also own a set of old transactions, outstanding loans, etc.

An entity may have any number of attributes, own any number of sets and belong to any number of sets. An entity can both own a set of a given name and belong to a set with the same name. These general facilities allow more specialized structures, such as trees and networks, to be expressed more or less trivially within the general framework of entities, attributes and sets. This is discussed in section 3.2. A more detailed discussion of the power of the EAS formalism can be found in (11).

During the systems analysis phase the design team should decide on the Entities, Attributes, and Sets that are needed to describe the status of the system. These should be documented, preferably in the manner shown below. Once this is done, event programs can be designed to alter the EAS status as necessary†.

Exhibit 1 documents the EAS structure of a system that keeps track of JOBs of different types. JOBs consist of TASKs which, in turn, may be made up of SUBTASKS. The responsibility for these JOBs is delegated to GROUPs which are subdivided into AREAs.

The EAS structure is documented in a columnar fashion. The first column contains the names of the entity types in the system. Each entity type is followed by a list of its attributes in column two. Columns three and four contain the names of the sets the entity type owns and belongs to respectively. To the right of column four is a space for comments.

The format of exhibit 1 is recommended for documenting database and main storage EAS structures but the structures have to be defined by translating this information into English-like definitions. We are, however, working on a facility that will allow the user to define EAS structures directly with a format similar to this.

3. THE EAS-E PROGRAMMING LANGUAGE

Since the status of an application or a system is described completely in terms of the entities, attributes, and sets in existence at any point in time, events can change status in only a limited number of ways. They can:

- Create or destroy entities.
- Assign or change attribute values.
- File or remove entities from sets.

Commands for the above actions have an English-like syntax. For example:

```
CREATE A TASK
DESTROY THE TASK CALLED T2
LET PRIORITY(TASK) = 5
FILE THIS TASK IN JOB.TASKS
REMOVE THE FIRST TASK FROM JOBS.TASKS
```

† The acronym EAS-E comes from Entities, Attributes, Sets and Events

Entity	Attribute	Owns	Belongs	Comment
JOB				
	JOB_NUMBER			Text variable
	JOB_CUSTOMER_NAME			Text variable
	JOB_DESCRIPTION			Text variable
		JOB_TASKS		Tasks of the job
TASK				
	TASK_AREA_NUMBER			Alpha variable
	TASK_DESCRIPTION			Text variable
	TASK_JOB_NUMBER			Text variable
	SUBTASK_OWNER			Identifier variable
		SUBTASKS		Tasks can both own
			SUBTASKS	and belong to subtasks
			JOB_TASKS	
			AREA_TASKS	
AREA				
	AREA_NAME			Text variable
	AREA_NUMBER			Alpha variable
	AREA_TYPE			Integer variable
		AREA_TASKS		Tasks of the area
			GROUP_AREAS	
GROUP				
	GROUP_NAME			Text variable
	GROUP_NUMBER			Alpha variable
		GROUP_AREAS		Areas of the group

Exhibit 1: An Example EAS Structure

Thus, EAS-E programs, which are collections of such commands, can be read like English prose. Note that the commands must conform to a predefined syntax, they cannot be written in free English, but the syntax has been designed to make their intentions clear and unambiguous.

We will illustrate features of the EAS-E language by examples† from the first application system developed using EAS-E: a rewrite and extension of the Workload Information System of Thomas J. Watson's Central Scientific Services (CSS). CSS consists of about 100 craftsmen who do model shop, glass work, electronics, etc., for Thomas. J. Watson's scientists and engineers. The old Workload Information System, written in PL/I and assembler, was difficult to modify or extend. The EAS-E version duplicated the function of the old system: it read the same weekly inputs and generated the same outputs. It achieved this with about one-fifth as much source code as the old system. It also showed an even greater, but difficult to quantify, advantage over the old system in terms of ease of modification and extension. The system has now been extended to accept certain inputs and provide certain outputs on-line rather than batch.

3.1 Understandable Syntax

Exhibit 2 is a routine from the CSS system whose function is, as it says, to start a new page and print a heading. This routine is called in the printing of several different reports. The START NEW PAGE statement is self-explanatory. The seven lines (including the blank seventh line) following the PRINT 7 LINES... statement are printed during run-time just as they appear in the source program, except that the first variable, PAGE.V (automatically maintained by EAS-E) is printed in place of the first grouping of ***s, the second variable TITLE is printed in place of the second groupings of *s, etc. Though the routine is without comment, its action should be clear to anyone.

The PRINT statement in exhibit 3, slightly simplified from its CSS version, prints 3 lines containing the specified variables and expressions in the places indicated in the three form lines (last 3 lines of the exhibit). These lines will print data under the headings of exhibit 3.

The example of a FIND statement reproduced in exhibit 4, including the IF ONE IS (NOT) FOUND statement that usually follows a FIND, finds the job in CSS_JOBS with the specified job number. In fact, the FIND statement

† The complete syntax is available in (8).

Exhibit 2:

```
ROUTINE TO START_NEW_PAGE_AND_PRINT_HEADING
START NEW PAGE
PRINT 7 LINES WITH PAGE.V, TITLE, SUBTITLE, DATE, AND WEEK THUS...
CSS Information System                                          Page ***
                    Central Scientific Services
     ************************************************
*****************************                          ********** **
CSS  CSS  DEPT  CHARGE ENTRY COMPL ESTIMT PRCDNG TOTAL  TIME   CUSTOMER NAME
AREA JOB  NUM     TO    DAY   DAY          WEEK   TIME RMNING  /AREA RESP.

RETURN   END
```

Exhibit 3:

```
 PRINT 3 LINES WITH TASK_AREA_NUMBER, JOB_NUMBER, JOB_DEPT_NUMBER,
JOB_PROJ_NUMBER, TASK_ENTRY_DATE, TASK_COMPL_DATE, TASK_ESTIMATED_HOURS,
INHOUSE_HOURS_FOR_WEEK, TASK_INHOUSE_HOURS+TASK_VENDOR_HOURS,
TASK_ESTIMATED_HOURS - TASK_INHOUSE_HOURS - TASK_VENDOR_HOURS,
JOB_CUSTOMER, JOB_DESCRIPTION, TASK_EST_VENDOR_HOURS,
VENDOR_HOURS_FOR_WEEK, TASK_VENDOR_HOURS, TASK_ASSIGNEE_NAME, AND STAR THUS
*** ***** ****  ****  ***  *** ****** ****** ****** ****** *************
************************************  ****** ****** ****** ************ *
-------------------------------------------------------------------------
```

Exhibit 4:

```
 FIND THE JOB IN CSS_JOBS WITH JOB_NUMBER = PROPOSED_JOB_NUMBER
   IF ONE IS FOUND...
      CALL REJECT(|JOB ALREADY EXISTS WITH SPECIFIED JOB NUMBER.|)
      RETURN
   ELSE...
```

permits much more generality than is illustrated here, while retaining its readability if entity, attribute and set names are chosen with care.

The meaning of the FIND statement should be clear even to a reader without training in EAS-E. In general we have tried to design the EAS-E syntax to be self-documenting. A number of people without EAS-E training have commented on the clarity of the EAS-E source program for CSS, though it has very few comments (but carefully chosen names).

3.2 Sets as Standard Data Structures

In an experiment conducted by one of the authors (10), subjects were asked to design a program to be implemented in a higher level language that would retrieve information from three files in response to certain specified types of queries. Each of the eight subjects designed a different data structure to hold the information in main storage. If the designs were to be implemented, programs would need to be written to initialize each data structure and manage the pointers for all the operations on it.

We argue that the set facilities in EAS-E can be used to provide a large variety of data structures. These structures can be used by the programmer without having to initialize them and maintain their pointers. This is taken care of by routines automatically generated by EAS-E. The programmer merely thinks in terms of FILE and REMOVE operations.

For example, a stack is merely a LIFO (last-in-last-out) set. Stack operations can thus be implemented by FILE and REMOVE statements on a LIFO set. Similarly, queues and pipelines are FIFO (first-in-first-out) sets. A tree can be composed by defining an entity called a NODE that owns a set called LINKS and also belongs to a set called LINKS. The tree can be generated by creating NODES and filing them into LINKS of other NODES.

4. INTEGRATED LANGUAGE

Traditionally, programming languages work with database objects by imbedding a database language within the host programming language. For example, a PL/I or COBOL program that works with IMS (4) contains explicit calls to

the database language for database functions. To work with System R (I) the program contains statements in a special database language called SQL. In these cases, the database language can fetch, create, delete and update database objects while the host language takes care of manipulating the objects in main storage.

EAS-E is organized quite differently. The same language can be used to work with database entities as well as main storage entities in an entirely equivalent manner. In general, database and main storage variables can appear anywhere they make logical sense in any statement -- READ, WRITE, LET, IF, FOR EACH, etc.

Integration of host and database language shifts the burden of interfacing the host language to the database language from the programmer to the EAS-E system. This eliminates a great deal of the non-problem-related code that clutters up traditional programs and makes them difficult to understand. To illustrate this reduction in coding, we compare a typical PL/I program that works with an IMS database and a PL/I program that works with SQL with equivalent programs in EAS-E.

4.1 Structure of a PL/I Program Working with IMS

We discuss below the structure of a PL/I batch program which retrieves data from a detail file to update a master data base. In EAS-E such a program would consist of only a few lines. Let us assume that the detail file consists of entities of type DETAIL. The update program would then have the following structure:

```
FOR EACH DETAIL DO
    FIND THE MASTER WITH • • •
    CALL UPDATE(DETAIL,MASTER)
LOOP
```

The FOR EACH statement brings each DETAIL entity in turn from the database. The FIND statement brings in the appropriate MASTER entity. Changes made to the database by the UPDATE routine are made permanent when the program ends.

To compile this program it would by necessary to precede it with the statement:

```
DATABASE ENTITIES INCLUDE MASTER AND DETAIL FROM • • •
```

This would cause the definitions of MASTER and DETAIL, which would also be stored in the database, to be brought in and used to generate the the object code.

In IMS database objects are called segments. The example program works with segments of type MASTER and DETAIL. These segments are stored in separate data bases. Access to IMS databases is provided by making calls to DL/I (Data Language I) to Get, Insert, Delete or Update a specific segment. The call has the structure:

```
CALL PLITDLI(parm-count, call-function, db-pcb-name, i/o-area [, ssa ... ]);
```

The parameters of the call are discussed in Exhibit 5 below. Exhibit 6 illustrates in outline form the fundamental parts in the structure of the PL/I batch program. Exhibit 7 is the explanation of the program shown in exhibit 6. Exhibits 5, 6 and 7 are all taken directly from (5)

The skeleton program displayed in exhibit 6 is comprised of three kinds of statements. First, there are the declarations that set up the parameters to be used in the calls to DL/I (2, 3, 6). These are necessitated by the relative inflexibility of the call. There is no need to set up such parameters in an EAS-E program.

The second set of statements sets up the communication with DL/I (I, 4, 5, I1, 12). The equivalent function is provided in an EAS-E program by the DATABASE ENTITIES INCLUDE statement. In addition to retrieving the definitions of the entities to be accessed at compile time it also causes the compiler to include in the object program the instructions required to establish contact with the database and to test the return codes from requests to the database manager.

The third part of the skeleton program consists of the actual calls to the database management system (7, 8, 9, 10). What is not illustrated is the control structure that must surround these calls to check the return codes and take the requisite actions. After an DETAIL is processed the program must jump back to the call that brings the next detail. If the last DETAIL has been processed it must terminate. Here again, EAS-E, being an integrated language, simplifies matters for the programmer. The FOR EACH statement fetches each entity of a given type (or, as we shall see later, entities in a given set) from the database. Main storage operations to be performed on the entity are specified in the body of the statement.

4.2 A PL/I Program Working With SQL

We now discuss a PL/I program working against System R (1,2) a more modern database management system. As we shall see, the difficulties of interfacing to the database management system have been mitigated but a certain amount of non-problem related code is still necessary. The PL/I-SQL program shown in exhibit 8 has 20 lines. The equivalent EAS-E program shown in exhibit 9 has 6 lines.

parm-count
The first parameter is the address of a four byte field containing the number of other parameters that are in the list.

call-function
The second parameter contains the address of a four character field that contains the DL/I code for the function to be performed.

db-pcb-name
The third parameter is the address of a program communication block (PCB) that is used for communication between DL/I and the application program. There is one PCB, which is contained within the PSB [Program Specification Block], for each data base being processed.

i/o-area
The I/O area address is the fourth parameter in the call statement. The I/O area is an area in the application program in to which DL/I puts a requested segment or from which DL/I takes a designated segment ... . The area must be as long as the longest segment to be processed. ... Segment data is always left justified within a common area. Because of the structuring of PL/I, i/o-area must be the name of a fixed length character string, an area, a level 1 in a structure, or an array. If the user wishes to deal with substructures or elements of an array, he should use the DEFINED or BASED attribute.

Example:

```
DECLARE 1 INPUT_AREA
          2 KEY CHAR(6),
          2 FIELD CHAR(84);
```

ssa
The addresses of one or more segment search arguments (SSAs) are the final parameters in the call statement. When an application programmer requests DL/I to perform data base functions, it is frequently necessary for him to identify a particular segment by its name and the names of all parent segments along the hierarchical path leading to the desired segment.

These values do not appear directly in the call statement argument provided to DL/I. Instead, an SSA name is given that points to an area in the user's program that contain the actual SSA values.

The SSA may consist of three elements: the segment name, the command code, and a segment qualification statement. The segment name provides DL/I with enough information to define the type of segment. The command code is optional and provides specification of functional variations applicable to the call function. ... The segment qualification statement is optional and contains information that DL/I uses to test the value of the segment's key or data fields within the data base to determine whether the segment meets the user's specifications.

A segment qualification statement is composed of three parts: a segment field name, a relational operator and a comparative value.

An *unqualified SSA* is built using a 9 byte area with the segment name occupying the leftmost 8 bytes and a blank in position 9 ... . [A *qualified SSA* consists of a field with the segment name in the leftmost eight bytes directly followed by a segment qualification statement.]

The qualification test is terminated as soon as a segment type that satisfies the qualification test is found in the data base. This procedure continues for all SSAs in a DL/I data base call until the desired segment is found.

Exhibit 5: Parameters in Calls to DL/I

```
REF.
 NO.
        /*                                                          */
        /*                     ENTRY POINT                          */
        /*                                                          */
  1     DLITPLI: PROCEDURE (DB_PTR_MAST,DB_PTR_DETAIL)
                 OPTIONS (MAIN);
        /*                                                          */
        /*                     DESCRIPTIVE STATEMENTS               */
        /*                                                          */
        DCL DB_PTR_MAST POINTER;
        DCL DB_PTR_DETAIL POINTER;
        DCL  FUNC_GU      CHAR(4)     INIT('GU  ');
        DCL  FUNC_GN      CHAR(4)     INIT('GN  ');
  2     DCL  FUNC_GHU     CHAR(4)     INIT('GHU ');
        DCL  FUNC_GHN     CHAR(4)     INIT('GHN ');
        DCL  FUNC_GNP     CHAR(4)     INIT('GNP ');
        DCL  FUNC_GHNP    CHAR(4)     INIT('GHNP');
        DCL  FUNC_ISRT    CHAR(4)     INIT('ISRT');
        DCL  FUNC_REPL    CHAR(4)     INIT('REPL');
        DCL  FUNC_DLET    CHAR(4)     INIT('DLET');
        DCL 1    QUAL_SSA             STATIC UNALIGNED,
  3             2   SEG_NAME         CHAR(8) INIT('ROOT    '),
                2   SEG_QUAL         CHAR(1) INIT('('),
                2   SEG_KEY_NAME     CHAR(8) INIT('KEY     '),
                2   SEG_OPR          CHAR(2) INIT(' ='),
                2   SEG_KEY_VALUE    CHAR(6) INIT('vvvvvv'),
                2   SEG_END_CHAR     CHAR(1) INIT(')');
        DCL 1    UNQUAL_SSA           STATIC UNALIGNED,
                2   SEG_NAME_U       CHAR(8) INIT('NAME    '),
                2   BLANK            CHAR(1) INIT(' ');
        DCL 1    MAST_SEG_IO_AREA,
  4             2 ---
                2 ---
                2 ---
        DCL 1    DET_SEG_IO_AREA,
                2 ---
                2 ---
                2 ---
        DCL 1    DB_PCB_MAST          BASED (DB_PTR_MAST),
                2   MAST_DB_NAME     CHAR(8),
                2   MAST_SEG_LEVEL   CHAR(2),
                2   MAST_STAT_CODE   CHAR(2),
  5             2   MAST_PROC_OPT    CHAR(4),
                2   FILLER           FIXED BINARY (31,0),
                2   MAST_SEG_NAME    CHAR(8),
                2   MAST_LEN_KFB     FIXED BINARY (31,0),
                2   MAST_NO_SENSEG   FIXED BINARY (31,0),
                2   MAST_KEY_FB      CHAR(*);
        DCL 1    DB_PCB_DETAIL        BASED (DB_PTR_DETAIL),
                2   DET_DB_NAME      CHAR(8),
                2   DET_SEG_LEVEL    CHAR(2),
                2   DET_STAT_CODE    CHAR(2),
                2   DET_PROC_OPT     CHAR(4),
                2   FILLER           FIXED BINARY (31,0),
                2   DET_SEG_NAME     CHAR(8),
                2   DET_LEN_KFB      FIXED BINARY (31,0),
                2   DET_NO_SENSEG    FIXED BINARY (31,0),
                2   DET_KEY_FB       CHAR(*);
        DCL     THREE     FIXED BINARY    (31,0)   INITIAL(3);
        DCL     FOUR      FIXED BINARY    (31,0)   INITIAL(4);
        DCL     FIVE      FIXED BINARY    (31,0)   INITIAL(5);
        DCL     SIX       FIXED BINARY    (31,0)   INITIAL(6);
```

Exhibit 6: Structure of a PL/I Program With DL/I Calls

```
        /*                                                          */
        /*   MAIN PART OF PL/I BATCH PROGRAM                        */
         •
 7           CALL PLITDLI(FOUR,FUNC_GU,DB_PCB_DETAIL,
             DET_SEG_IO_AREA,QUAL_SSA);
         •
 8           CALL PLITDLI(FOUR,FUNC_GHU,DB_PCB_MAST,
             MAST_SEG_IO_AREA,QUAL_SSA);
         •
 9           CALL PLITDLI(THREE,FUNC_GHN,DB_PCB_MAST,
             MAST_SEG_IO_AREA);
         •
10           CALL PLITDLI(THREE,FUNC_REPL,DB_PCB_MAST,
             MAST_SEG_IO_AREA);
         •
11           RETURN;
             END DLITPLI;
-------------------------------------------------------------------
12           PL/I LANGUAGE INTERFACE
```

Exhibit 6 (Contd.): Structure of a PL/I Program With DL/I Calls

System R supports relational data structures. Relations can be considered to be tables with each tuple (row) of a relation being a collection of attributes in a given order. The example application program shown in exhibit 8 is taken from (2). It updates the salary of some of the employees whose records are contained in a relation called EMP. Each row of EMP refers to a particular employee and stores the EMPNO, DEPT, SAL, RATING and possibly other attributes. Statements that start with "$" are SQL statements.

The first line of the program contains its name and specifies that it has one argument: XDEPT. The SQL statement on line 2 declares the main storage variables XEMPNO, XDEPT, XSAL, XRATING and XRAISE. Subsequently these variables, referred to as $XEMPNO, $XDEPT, etc., can be used in SQL statements.

The SQL statement on lines 3 to 7 defines a cursor called C1 which consists of the EMPNO, SAL and RATING attributes of tuples in the EMP relation that meet the condition DEPT = $XDEPT. The remaining part of the program processes C1, one tuple at a time, by moving the EMPNO, SAL and RATING attributes into $XEMPNO, $XSAL and $XRATING respectively. The $OPEN statement on line 8 readies the cursor for processing and binds the value of $XDEPT. The DO WHILE on line 8 starts a perpetual loop whose scope is terminated by the END statement on line 17. Within this loop the $FETCH statement moves the attribute values from the current tuple of the cursor into the appropriate main storage variables. The raise is then computed and stored in $XRAISE. The $UPDATE statement on line 14 stores the new value of the SAL attribute in the current tuple of C1.

Line 11 tests the return code from the $FETCH and jumps out of the loop when the cursor is exhausted. The $CLOSE statement closes C1 and the program ends.

4.3 Equivalent Programs in EAS-E

The function performed by the program in Exhibit 8 can be performed by the single EAS-E statement:

```
FOR EACH EMPLOYEE WITH DEPT = XDEPT
  LET SALARY = • • •
```

This brings each employee in turn from the database into main storage and tests whether its DEPT attribute equals XDEPT. If this is so, it updates its salary as specified. The database modifications are made permanent when the program ends.

For compilation, the above statement must be preceded by:

```
DATABASE ENTITIES INCLUDE EMPLOYEE FROM • • •
```

Bringing all the entities of a given type into main storage and processing those that meet given conditions is exactly equivalent to processing a relation in a relational database. In a network database we can be more efficient. If the selection criterion is one that is commonly used, such as employees within a department, then the EMPLOYEE entities should be filed into sets owned by the DEPARTMENT entities. With this structure, only the EMPLOYEEs in the particular department are brought into main storage.

The following explanation relates to the reference numbers along the left side of [Exhibit 6].

1. This is the main entry point to a PL/I batch program. After the DL/I control program has loaded and relocated the PSB for the program, it gives control to this entry point. The PSB contains all the PCBs used by the program. The entry point statement of the batch program must contain the same number of names in the same sequence as there are PCBs in the PSB.

2. Each area defines one of the call functions used by the PL/I batch program. Each character string is defined as 4 alphameric characters, with a value assigned for each function ... . Other constants may be defined in the same manner. Standard definitions could be stored in a source library and included using a %INCLUDE statement.

3. A structure declaration defines each SSA used by the problem program. The unaligned attribute is required for SSA data interchange with DL/I. The SSA character string must reside contiguously in storage. Assignment of variables to key values, for example, could result in the construction of an invalid SSA if the key value has the aligned attribute.

   A separate SSA structure is required for each segment type accessed by the program because the key-value fields should be different. Once the fields other than key-value are initialized, they should not have to be altered.

   A 9 byte area should be reserved for use as an unqualified SSA. Before issuing an unqualified call, a segment name is moved into this field.

4. The segment I/O areas are defined as structures.

5. One level 1 declarative ... describes as a structure the data base PCB entry for each input or output data base. It is through this description that a PL/I program may access the status codes after a DL/I call.

6. This statement is used to identify a binary number (fullword) that represents the parameter count of a call to DL/I. ...

7. and 8. These are typical calls to retrieve data from a data base using a qualified SSA.

   Prior to execution of the call the SEG__KEY__VALUE field of the SSA must be initialized if a fully qualified SSA is required. For a call using an unqualified SSA, the segment name field must be moved to one of the 9-byte UNQUAL__SSA areas.

   Immediately following the call the status code field of the PCB must be checked to determine the results of the call. ...

9. This is a typical call to retrieve data from a data base using no SSA. This call is also a HOLD call for subsequent delete or replace operation.

10. This call is used to replace data in a data base with data from a PL/I batch program.

11. This RETURN statement causes the batch program to return control to DL/I.

12. [A language interface module must be link-edited to the program to provide a common interface to DL/I. The specific module used and the procedure for link-editing depend on the operating environment.]

Exhibit 7: Comments on The PL/I-DL/I Program Structure

```
 1   PAYRAISE: PROC(XDEPT);

 2       $DCL (XEMPNO, XDEPT, XSAL, XRATING, XRAISE);
 3       $LET C1 BE
 4               SELECT    EMPNO, SAL, RATING
 5               INTO      $XEMPNO, $XSAL, $XRATING
 6               FROM      EMP
 7               WHERE     DEPT = $XDEPT;
 8       $OPEN C1;
 9       DO WHILE ('1' B);
10           $FETCH C1;
11           IF SYR_CODE ¬= 0 THEN GO TO WRAPUP;
12           /* COMPUTE XRAISE BASED ON
13               XSAL AND XRATING */
14           $UPDATE EMP
15               SET SAL = SAL + $XRAISE
16               WHERE CURRENT OF C1;
17       END;
18   WRAPUP;
19   $CLOSE C1;

20   END PAYRAISE;
```

Exhibit 8: An Example PL/I Program With SQL statements

Exhibit 9 shows an EAS-E subroutine that updates the salary of all employees in a given department. It uses a FOR EACH statement which is a little different from the one used in the above example: it processes all the entities in a set rather than all the entities of a certain type.

Exhibit 9:

```
ROUTINE PAYRAISE(XDEPT)
DEFINE XDEPT AS A REFERENCE VARIABLE
FOR EACH EMPLOYEE IN EMPLOYEES(XDEPT)
   LET SALARY = • • •
RETURN
END
```

For compilation, the above subroutine would have to be preceded by

DATABASE ENTITIES INCLUDE EMPLOYEE AND DEPARTMENT FROM •••

Let us now discuss the actions caused by the above program in more detail. The DATABASE ENTITIES INCLUDE statement specifies that the program will be working with the database entities EMPLOYEE and DEPARTMENT. In addition, it declares EMPLOYEE and DEPARTMENT as reference variables which can be used to point to EMPLOYEE and DEPARTMENT entities in main storage. During compilation the definitions of EMPLOYEE and DEPARTMENT are brought from the database and the information contained in them is used to generate object code that can manipulate their attributes in main storage.

The DEFINE statement defines XDEPT as a local reference variable. When the routine is called, XDEPT must refer to a database entity of type DEPARTMENT. The next statement, FOR EACH EMPLOYEE IN EMPLOYEES(XDEPT), instructs the compiled EAS-E program to have the reference variable "EMPLOYEE" point in turn to each member of the set called EMPLOYEES that is owned by the DEPARTMENT pointed to by XDEPT. This statement could be followed by an arbitrary number of selection phrases (such as the WITH... phrase in the example above) that would instruct the executing program to test the EMPLOYEE and make sure that it met specified conditions. For each EMPLOYEE thus selected, the program would update the SALARY attribute in the manner specified.

5. SUMMARY

This paper has discussed the Entity, Attribute and Set model and described a simple method of documenting the EAS status of a system or application. It has also discussed the EAS-E language and compared its facilities with other languages from the point of view of compactness and readability. We have shown that EAS-E programs are much shorter than equivalent programs in other languages and contain much less non-problem-related code. Since the powerful, English-like commands in EAS-E programs accurately mirror the steps taken by the programmer to solve the problem, EAS-E can be thought of as executable documentation.

6. REFERENCES

1. Astrahan, M.M., Blasgen, M.W., Chamberlin, D.D., Eswaran, K.P., Gray, J.N., Griffiths, P.P, King, W.F., Lorie, R.A., McJones., P.R., Mehl, J.W., Putzolu, G.R., Traiger, I.L., Wade, B.W., and Watson V., System R: A Relational Approach to Database Management., *ACM Trans. Database Syst., 1,* 2, (June 1976), 97-137

2. Blasgen, M.W., System R: An Experimental Relational Database Management System, Lecture Notes. IBM Research, San Jose, May 19, 1979.

3. CODASYL *Data Base Task Group Report,* Available from ACM, New York, NY, April 1971.

4. IBM Corporation, *Information Management System/360, General Information Manual,* Prog Prod 5734-XX6, GH20-0765 06061

5. IBM Corporation, *Data Language/I Disk Operating System/Virtual Storage: Application Programming Reference Manual,* Prog Prod 5746-XX1, SH12-5411-4

6. IBM Corporation, *Virtual Machine/370 Display Management System for CMS: Guide and Reference,* Program Number 5748-XXB File No. 5370-39 SC24-5198-0

7. Kiviat, P.J., Villanueva, R., & Markowitz, H.M., *The SIMSCRIPT II Programming Language.,* Prentice Hall, Englewood Cliffs, NJ, 1969

8. Malhotra, A, Markowitz, H.M., & Pazel, D.P., The EAS-E Programming Language., RC 8935, IBM T. J. Watson Research Center, Yorktown Hts., NY 10598.

9. Malhotra, A, Markowitz, H.M., & Pazel, D.P., EAS-E: An Integrated Approach to Application Development, RC 8457, IBM T. J. Watson Research Center, Yorktown Hts., NY 10598. *ACM Trans. Database Syst.,* To appear.

10. Malhotra, A., Thomas, J.C., Carroll, J.M. & Miller, L.A., Cognitive Processes in Design, *Intl. Jrnl. Man-Machine Studies. 12* 2, (Feb. 1980), 119-140

11. Markowitz, H.M., SIMSCRIPT, *Encyclopedia of Computer Science and Technology,* Belzer, J., Holzman, A.G. and Kent, A. (eds.), Marcel Dekker, New York, 1979, pp 79-136. Also RC 6811, IBM T. J. Watson Research Center, Yorktown Hts., NY 10598

12. Markowitz, H.M., Malhotra, A, & Pazel, D.P., The ER and EAS Formalisms for System Modeling, and the EAS-E Language, RC 8802, IBM T. J. Watson Research Center, Yorktown Hts., NY 10598.

ADD: An Automated Tool For Program Design and Documentation

T.C. Ting*

Worcester Polytechnic Institute

An approach which integrates the activities of software design and documentation is proposed, described, and discussed. An automated tool called ADD which uses a data dictionary system is suggested to support this approach. The unified approach not only offers solutions to some of the important documentation problems, but it provides a structured means for better program design and coding. Program design process is enhanced and guided by a structured "design template". Program design documents are generated automatically to serve as "blueprints" for programming. The use of a "program coding template" provides a structure for coding. Program module interface conditions are automatically generated and controlled from the design specifications. Program modules are tested by using the predesigned and stored test data to certify their correctness.

The structure of the tool is illustrated. How the automated tool may be used and the benefits of such an automated tool are discussed.

Keywords: Automated tools, program design, program documentation, program document standardization, program testing, software engineering.

*The author is also a part-time computer scientist at the Institute for Computer Sciences and Technology, the National Bureau of Standards. This paper was not prepared while on official duty and it does not reflect any official position of the National Bureau of Standards.

1. INTRODUCTION

Studies of large computer software systems have identified several major problems. One of the most serious concerns is the sad state of computer software documentation (GAO 1974). The literature reports many suggestions and recommendations on the contents, methods, and formats of software documentation (Brewer 1976, Gey 1973, Gass 1979). Standardized software documentation procedures and formats have been adopted by many organizations including the Federal Government (NBS 1976, 1980). However, the problem remains.

The author is a strong believer that in order to yield a good software product, the software documentation activities must be integrated into the whole software development process. Program documentation is an active part of program development. It should not be treated as a passive task of simply recapturing the descriptions of an already developed program. Good program design leads to good documentation. Good documentation contributes to good design.

An organized and structured program design, together with straightforward program implementation techniques, can immensely ease the documentation task (Gey 1976). Dijkstra stated that good programming recognizes how to avoid unmanageable complexity (Dijkstra 1975). A well-structured top-down design decomposes a large and complicated problem into multiple units of comprehensible and manageable subproblems and sub-subproblems. These sub-divided units can be organized into program modules. These modules are the basic building blocks for the program and they can be individually designed, implemented, and tested. The proposed approach with an automated tool supports the structured program design and implementation.

This paper reports and discusses the concepts of the automated program design and documentation approach. Four types of software documents are necessary for a software system. These documents are prepared for management, users, operational personnel, and programming

analysts. However, the documentation discussed in this paper concerns only the type of documents for program analysts. These documents are used in assisting program design, implementation, and maintenance.

An automated program design and documentation tool called ADD, Automated Design and Documentation, is suggested and illustrated. ADD uses a data dictionary and program design and coding templates. It provides a means for the designer to have an organized and structured approach for program design. The results of this approach not only provide a set of comprehensive program documents, but produce a good program design.

ADD automatically generates program structural and flow diagrams, and individual program module description sheets. These documents can help the designer to verify the design, and they serve as "blueprints" for programming. A program coding template is used to help in setting up a structure for programming. Important program module interface conditions such as input and output variables are automatically generated from the design which is stored in the data dictionary. The validity of the implementation is checked automatically and any conflict must be resolved between the designer and programmer. Test data provided by the designer are part of the program documentation and they are used to certify the program during the program testing and maintenance stages.

One of the most difficult problems of software documentation is the ever-changing environment of a software system. The high rate of software maintenance activities and program patches make any existing software document quickly out-of-date. One of the major benefits of ADD is to reduce this problem. The organization and structure of the tool are presented and discussed. The benefits of the proposed approach and the automated tool, ADD, are discussed.

2. PROBLEMS OF SOFTWARE DOCUMENTATION

Documentation is an area of the software engineering field where the professional program analyst is not very competent. Almost no one has had the actual opportunity to learn to document. First of all, university computer science curricula have never paid any serious attention to program documentation. A large number of software packages have been produced by universities from student and faculty research and development projects. These software products have greatly contributed to computer software advancements. However, software documents for these products are almost non-existent or too poor to be useful. Students are usually taught to code. Emphasis is often misplaced on the programming language instead of design and programming techniques. Not much stress is laid on program design and documentation. Due to the classroom environment, usually only small, simple and trivial program exercises can be assigned. These assignments often do not justify applying a rigid program development methodology. Students usually work alone with no requirement to cooperate with others. In many cases, instructors discourage or even prohibit students working cooperatively on programming assignments. Student programs are often required to run only once for some arbitrarily selected sets of data for grading purposes. Under such circumstances, no incentive is given to the design and to the documentation of the program.

Documentation is usually treated as an add-on task to the software development project. Software development staff are often assigned to a new project immediately after the program is running and before the documentation has been completed. Frequently, time provided for documentation is not sufficient. Many program documents are unverified as to their completeness and correctness. The importance of the documentation is considered only as an afterthought whenever some difficulties have occurred during the program maintenance stage. It is very difficult to convince the management and project development staff that without the completion of program documentation the project has not been completed. The problem is compounded by the fact that many software development methodologies consider documentation activities separately from the program development. Documentation is usually done after the program has been completed.

The most difficult problem of program documentation is the ever-changing software environment. Continuously changing program requirements or computing facilities makes it very tough and laborious for software maintenance personnel to keep software running smoothly. Frequently, a program has to be modified for whatever the reasons may be. The

numerous program patches make any existing program document quickly out-of-date. The document is often not immediately updated. It is very difficult for the busy maintenance staff who is under pressure to fix the program to consider concurrently changing the program document. Poor documentation causes the maintenance personnel to be even busier.

The continuing proliferation of novel hardware devices, programming languages, and additional applications complicates further the task of software design and documentation. It is difficult to formulate any documentation standard and procedure that can really work.

Automation of program design and documentation offers some possible solutions to some of the above mentioned problems.

3. PURPOSES AND TYPES OF SOFTWARE DOCUMENTATION

Generally speaking, four types of software documentations are needed for different purposes (Gass, 1981). They are prepared for users, operational personnel, management, and program analysts.

User documentation is a medium which provides for communications between the user and software designer. It provides users with instructions for using the system, such as descriptions on the contents and forms of input and output of the software. Ideally, it should provide answers to all of the user's questions concerning the use of the system.

Operational documentation specifies requirements and steps necessary for operating the software. It is prepared for the operational personnel including data preparation, data entry, security and integrity checking, just to name a few.

Software documentation for management is a general statement of the conceptual design of the proposed system (prepared for management decision-making). It specifies the functional and performance requirements of the system and resources required for the development and later operation of the system such as personnel, financial, and computing resources. The final document is the basis for the development of the system.

This paper discusses mainly the software documentation for the program analyst. We will refer to this type of documentation as program documentation. It describes the detailed design and implementation of the program. It provides technical descriptions for the program. It has a set of functional specifications. It defines all input and output for the whole program as well as its subprograms. It lists the test data and their expected results. It provides detailed descriptions on all data files and records. It specifies the environment under which the program is to operate.

Program documentation assists the technical staff to develop and to maintain programs. It is a record of the program design and implementation, and it is a technical reference to the program.

4. SOFTWARE DEVELOPMENT AND DOCUMENTATION

Software documentation activities are the integral part of software development. Documentation activities are involved in every stage of the software development life cycle: problem definition, system design, programming, testing, and maintenance. Software documentation keeps a formal record of the results of each major software development step.

Good software documentation imposes a structured approach on the problem definition. It helps in defining the problem in terms of the input, output, and functions of the program. It keeps track of the process of problem decomposition during the design phase. It helps to organize the multiple levels of subproblems into program modules. It offers a structure for systematic design of each program module including the functional specification, input and output variables, data files, the logical steps to transform from input to the desired output, and the necessary testing data and their expected results.

Software documents generated during the design phase are, in fact, the "blueprints"

for coding. The program listings are the program implementation documents which describe the actual programs in a chosen programming language.

Program design and implementation documents are the natural products of the software development life cycle. Program documentation enhances the program development process for better design and better implementation of the program, when the emphasis is on the design but not on coding. Software development includes software documentation. Software documentation is an integral and inseparable part of the software development task.

During the software maintenance stage, program documents are obviously essential. Any modification made to the program must immediately be recorded into the document so that the document is always an accurate and correct description of the program.

5. AUTOMATION OF PROGRAM DOCUMENTATION

The software profession has successfully automated many systems for others. It has not, however, been too successful for itself. Program documentation is one of the areas in which automation may be fruitful. The software profession needs automated documentation tools for enhancing software development. Among the four types of software documentation, program documentation is probably the best candidate for immediate automation.

This section discusses the proposed approach and ADD. The overall organization and involvement of ADD is illustrated in Figure 1.

5.1 Use of Data Dictionary System for Recording Program Design Specifications

A data dictionary system has demonstrated its usability in data management (Lefkowitz, 1977). This paper presents a new approach by using the data dictionary system in program design and documentation. In a top-down program design process, the problem to be solved by the proposed system is functionally decomposed into smaller subproblems. These decomposed units are organized into many interrelated program modules for implementation. Two types of modules are involved: terminal and drive modules. These modules are hierarchically organized in a program tree diagram which expresses the hierarchical relationship of modules by functional decomposition. Modules can also be represented in flow diagrams to show the process flow pattern of the program. These program modules are the basic building blocks for the program.

For each program module, the designer analyzes the functional specifications, and determines the input and output variables, the major logical steps to transform from input to output, the names of other modules called by this module for performing the functions, the test data and their expected results. A data dictionary system can be used to store these design specifications. The following categories of information are stored in the data dictionary:

Name and type of module
Functional specifications
Input and output variables
Names and descriptions of data files
Names and other program modules called by this module
Major logical steps of the process
Test data and their expected results

5.2 A Program Design Template

A program design template is proposed to assist the designer for entering program module design specification. It provides a structure so that the designer can proceed with the design in an organized and systematic manner. It helps the designer to completely specify each program module area by area. Areas to be specified are discussed below.

5.2.1 Name and Type of Module

The name of the module is assigned by the designer using the established naming

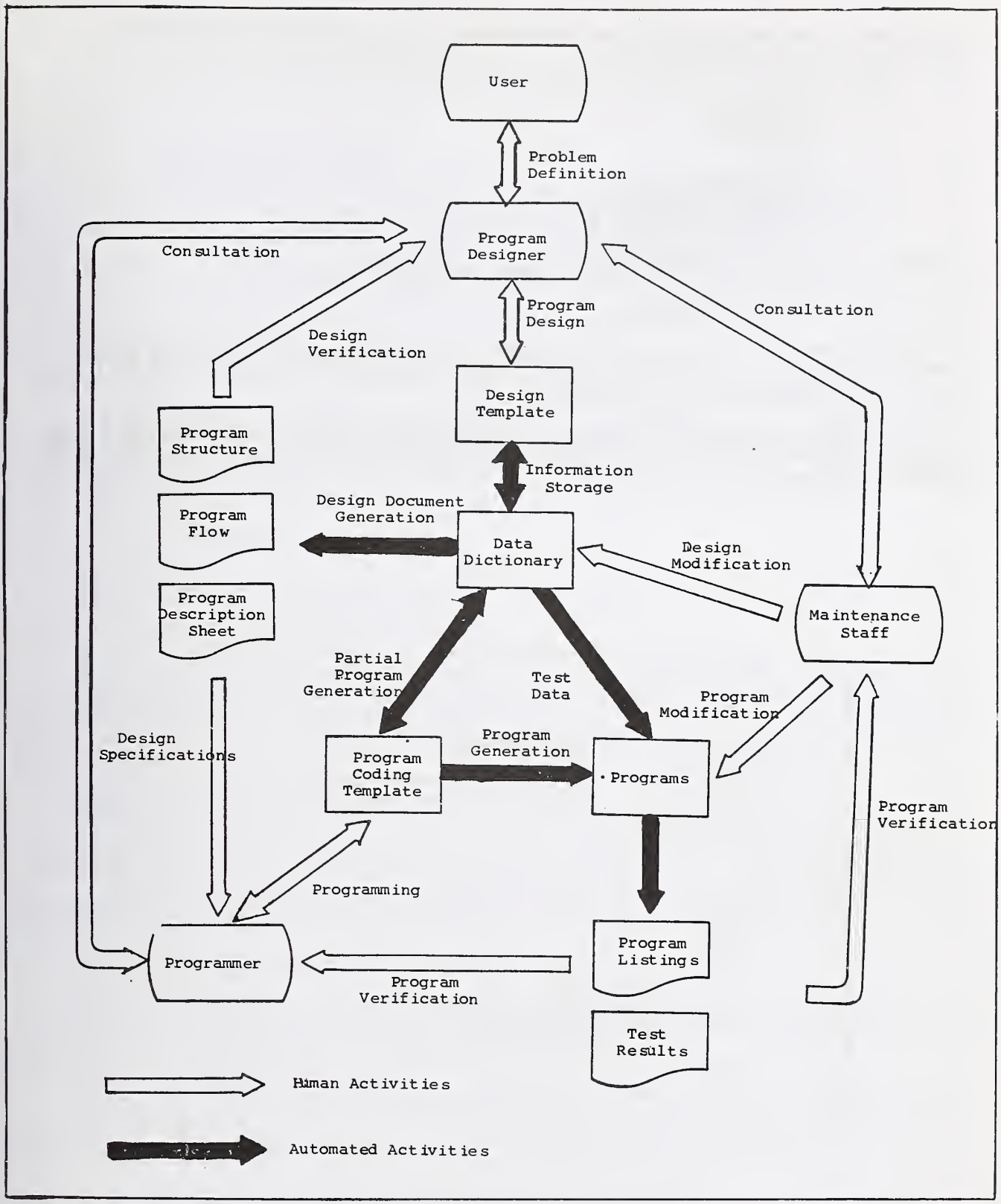


Figure 1. -- The Organization of ADD.

convention or standard adopted by the organization. The type of module is specified, and it may be one of the following:

A block
A PARAGRAPH
A PROCEDURE
A Macro
A REMOTE block
A function subprogram
A subroutine subprogram

The above is an example and it is by no means a complete list.

5.2.2 Functional Specifications

The functions to be performed by the module should be described in English or in mathematical logic. It is desirable that a single, meaningful, but not too trivial, function is to be performed by a program module.

The designer must specify the major processing steps by using the pseudo code. Only the main ideas of approaches are required. An example of the functional specification is provided in Figure 2.

```
FINDING ROOTS OF A QUADRATIC EQUATION.

INPUT VARIABLES ARE A,B, AND C.  THE RESULTS MAY BE:
1.  NO ROOTS
2.  ONE REAL ROOT
3.  TWO REAL ROOTS
4.  TWO COMPLEX ROOTS


MAJOR STEPS:

      INPUT

      IF A=0
        THEN IF B=0
               THEN NO ROOT
               ELSE ONE ROOT   (ONERT)
        ELSE IF B**2 LESS THAN 4AC
               THEN TWO ROOTS (TWORT)
               ELSE TWO COMPLEX ROOTS   (COMPRT)

      OUTPUT
```

Figure 2. -- An Example of the Functional Specification.

5.2.3 Input and Output Variables

For each program module, the designer must clearly specify the input and output variables. This is one of the most important module-to-module interface conditions. The following information must be provided:

Name of each variable.
Type of each variable.
Order of the input and output variables appears on the argument list, a common block, or an I/O data structure. If a named common block is used the name of the block, along with names of variables in desired order, must be specified.
Initial and range of values, if any, for each input and output variable.
Coding scheme, if any, for each input and output variable.

5.2.4 File Names and File Descriptions

Data files commonly accessed by different program modules must have their file descriptions stored in the file description modules in the data dictionary. (File modules are identified by their respected file names.) Each file description module includes detailed information on the file name, file organization, file protection methods, data records and their their descriptions, etc.

5.2.5 Other Modules Called by the Module

The designer must specify the names of those modules which are called by this module.

5.2.6 Test Data and Their Expected Results

Based on the functional specifications, the designer should design a complete set of test data. For each data group, the expected results should be provided. Additional testing conditions, if any, which are necessary for testing the robustness of the program module should also be provided. The error handling conditions or error return codes should be specified.

5.3 Automatic Generation of Program Design Documents

Information of each program module stored in the data dictionary can be automatically scanned to produce a set of program structural and flow diagrams. Individual program module description sheets can also be generated. These software design documents are the bases for programming. They can be treated as "blueprints" to the program.

5.3.1 Generation of Program Structural Diagram

The tree-like program structural diagram can be generated by scanning the names of modules called by other modules. The designer may use this diagram to verify the functional decomposition of the program.

The designer can modify the design. An updated program structural diagram can be regenerated without much difficulty. An example of the structural diagram is provided in Figure 3.

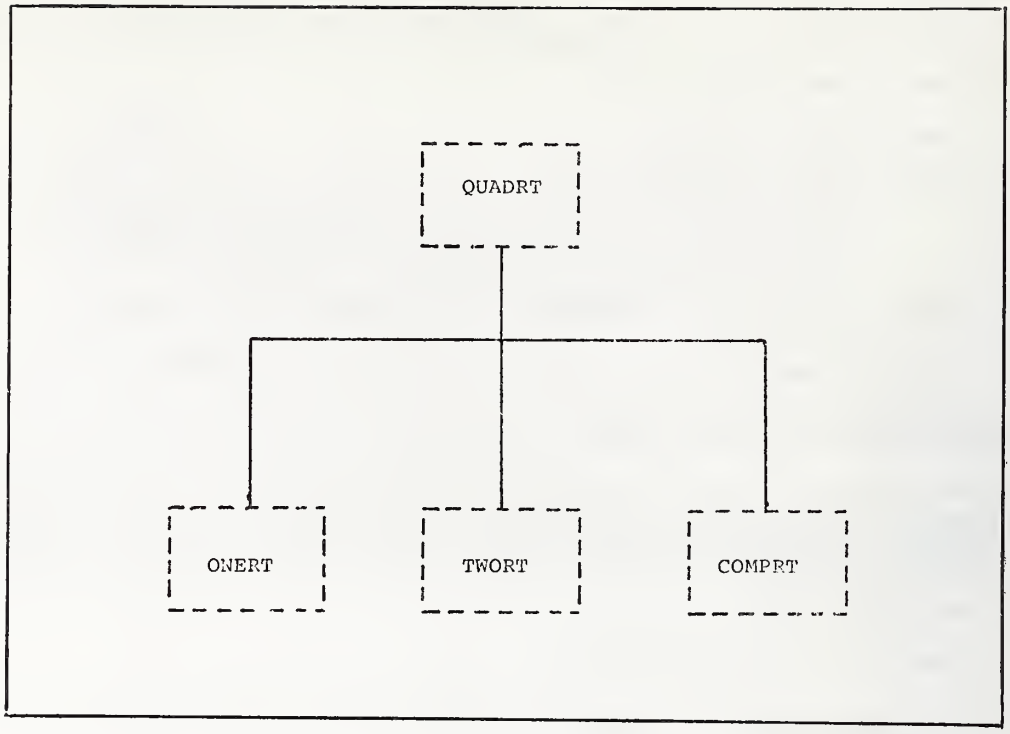


Figure 3. --- An Example of the
Generated Structural Diagram.

5.3.2 Generation of Program Flow Diagrams

Program flow diagram for each program module can be generated by scanning the pseudo code. Pseudo code consists of only two types of statements: decision and processing statements. It is not too difficult to develop a software package which can generate flow diagrams from pseudo code.

Flow diagrams can describe processing logic at different levels of abstraction. At the top of the program structure, the flow diagram reflects the function of the whole program in a highly abstract form. It may contain only several simple decisions for activating several major sub-program modules. However, at the lowest level, the processing logic of a small terminal module is described. Such a module should have only a simple function to perform. The generation of such a flow diagram may not even be necessary. However, it should not be difficult to produce such a diagram. In a well-structured and modularized design, if such a logic at the terminal module level is too complicated to describe, further decomposition is then necessary. An example of the flow diagram is illustrated in Figure 4.

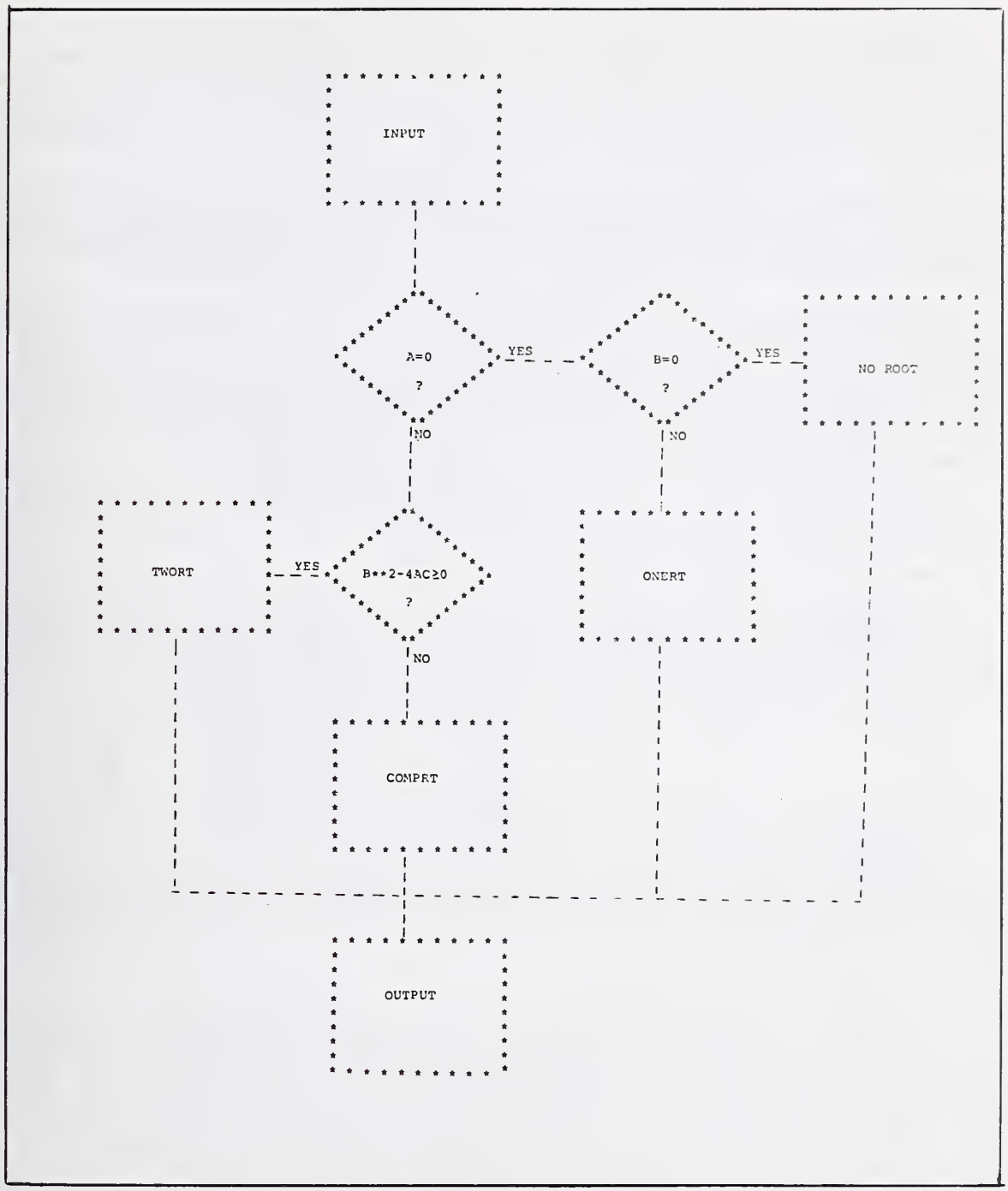


Figure 4. -- An Example of the
Generated Flow Diagram.

To flowchart or not to flowchart is a controversial issue. Some people deny the need for flowcharting, while others may consider flow diagrams necessary. It may be a good rule of thumb that at the terminal module level if it didn't require a flow diagram to code the module, it doesn't need a flow diagram. However, at the higher level, it needs a flowchart to keep track of what the designer is doing. The logical sequence of program modules must be made clear so that correct assumptions are made.

Flow diagrams or pseudo codes are useful to the designer. The logical processing sequences can be verified without worrying about the details in actual programming. These automatically generated diagrams help the designer to carry out the top-down design of the program. The designer need not consider the burden of producing these diagrams.

Program flow diagrams can assist the programmer in coding the program module into whatever the chosen programming language may be. When coding at the terminal module level, the programmer may exercise his or her own discretion in the implementation of the function. However, at the higher level, the drive module may require the process to follow a certain sequence as designed, since several modules may be involved to perform the function in a particular order. These program modules may be implemented by several programmers, and they must work according to the design.

The flow diagram helps the program maintenance staff. They can use these diagrams to understand on what the coded program listings are based.

5.4 Generation of Program Module Description Sheets

Program module description sheets can be automatically generated by simply printing out the sorted information in a preselected format. These description sheets can be used along with the structural and flow diagrams to provide detailed specifications for each program module. Some of the information described below have particular significance.

An example of the program module description sheet is provided in Figure 5.

5.4.1 Interface Conditions Between Program Modules

The module-to-module interface conditions can be clearly specified. On each program module description sheet, the input and output, the modules called, and the data files are detailed. This information is vitally important to the program development, especially when a large program is involved.

5.4.2 Program Testing and Test Data

To insure that each module performs as it is expected, a well-designed test data set must be provided. These data are developed by the designer. They are used to check the program implementation. During the maintenance phase, after each modification of the program, all modules involved must be tested by using these pre-designed and stored test data. These tests can insure to some degree that the modification is valid and no additional bugs have been introduced. This is obviously an important task. However, such a task has not been handled very well by many data processing organizations. Often, several bugs will be introduced after a simple modification to the program. New test data can be introduced if necessary, and they should be stored with the other test data in the data dictionary.

5.5 Other Design Aids

Other design aids such as cross references of various kinds can also be generated from the data dictionary. Details of this area have not been carefully investigated at this time. They will be reported in our forthcoming papers.

NAME OF THE MODULE: QUADRT

TYPE OF THE MODULE: SUBROUTINE

SPECIFICATIONS:

FIND ROOTS OF A QUAURATIC EQUATION

INPUT VARIABLES ARE A,B,C. THE RESULTS MAY BE:
1. NO ROOTS
2. ONE REAL ROOT
3. TWO REAL ROOTS
4. TWO COMPLEX ROOTS

MAJOR STEPS:

```
INPUT

IF A=0
  THEN IF B=0
         THEN NO ROOT
         ELSE ONE ROOT  (ONERT)
  ELSE IF B**2 LESS THAN 4AC
         THEN TWO ROOTS  (TWORT)
         ELSE COMPLEX ROOTS  (COMPRT)

OUTPUT
```

INPUT AND OUTPUT VARIABLES:

VARIABLE NAME	DESCRIPTIONS	TYPE	VALUE INIT	VALUE MIN	VALUE MAX	CODING SCHEME
A	INPUT VARIABLE	REAL				
B	INPUT VARIABLE	REAL				
C	INPUT VARIABLE	REAL				
RT	ROOT OF BX+C=0	REAL				
RT1	REAL ROOT ONE	REAL				
RT2	REAL ROOT TWO	REAL				
IM1	IMAGINARY ONE	REAL				
IM2	IMAGINARY TWO	REAL				
IND	RESULT INDICATOR	INTEGER				0=NO ROOT 1=ONE ROOT 2=TWO ROOTS 3=COMPLEX

DATA FILES:

OTHER MODULE CALLED:

ONERT, TWORT, COMPRT

TEST DATA AND EXPECTED RESULTS:

A	B	C	RT	RT1	RT2	IM1	IM2	IND
0.0	-5.0	2.0		-4.25	-5.75			2
4.0	4.0	2.0		-0.5	-0.5	1.0	1.0	3
0.0	2.0	4.0	-.05					1
0.0	0.0	5.0						0

Figure 5. -- An Example of the Program Module Description Sheet.

5.6 A Program Coding Template

A program coding template is suggested for use by the programmer in the implementation of the program. The program coding template can be implemented within the "Editor". Different types of program coding templates are necessary for different programming languages. The template is designed to assist the programming staff to avhieve a better structured and better documented program listing. Furthermore, program module interface conditions can be generated automatically from the design to allow better control and better implementation.

The template is activated by the programmer at the time when the program is to be created. It will ask the programmer to identify the program and program module. Basic information concerning the program module will then be obtained automatically from the data dictionary. The information includes the following:

Name of the module.
Specifications of the module. This information will be placed as the sequences of comment statements at the beginning of the program listing.
Input and output variables. An argument list, or any named or unnamed COMMON block, which contains the variables in their preselected names and order, is automatically generated. All declarative statements necessary to declare the input and output variables according to their types and initial values are generated. A set of comment statements which describe input and output variables is also generated.
File names and descriptions. The names of the files to be accessed by the module and their file descriptions are provided. The file description can be obtained from the file description module in the data dictionary.
A list of other modules called by this module.
A list of references which indicates where the design documentations of the module can be obtained. The design specifications of the module may be requested by the programmer from the data dictionary. This includes program structural diagram or the portion of the diagram, and flow diagram of the module. The descriptions of those modules which are called by this module can also be provided.

The programming coding template automatically sets up the basic structure of the program listing. Important information concerning the module interface are provided automatically. The vital module interface conditions are controlled and documented by the template according to the design.

An example of the editing process is illustrated in Figure 6.

6. ADVANTAGES OF USING ADD

The use of ADD has several advantages. It integrates documentation with program development activities to produce better designed, structured, and documented programs.

6.1 Integration of Program Design and Documentation Activities

The use of ADD insures that the program documentation and program design activities are inseparable. ADD provides a structured and systematic approach to design and implements the program. It assists the designer to completely specify the necessary design parameters. It provides an automated tool for the designer to verify and to refine the design. It helps the designer to organize the design works and to document the final design.

ADD provides the programming staff with the basic program structure and basic design specifications for implementing the program. The important program module-to-module interface conditions are generated automatically from the design. Other design information is referenced and documented within the program listing. The automatically generated program documents provide a complete set of design specifications which can be used as "blueprints".

The pre-selected test data not only are well designed, but they are permanently stored for use whenever a program test is required.

```
      SUBROUTINE QUADRT(A,B,C,RT,RT1,RT2,IM1,IM2,IND)
C*************************************************************
C
C     FIND ROOTS OF QUAURATIC EQUATION
C
C     INPUT VARIABLES ARE A,B,C.  THE RESULTS MAY BE:
C     1. NO ROOTS.
C     2. ONE REAL ROOT.
C     3. TWO REAL ROOTS.
C     4. TWO COMPLEX ROOTS.
C
C*************************************************************
C
C     A  INPUT VARIABLE
C     B  INPUT VARIABLE
C     C  INPUT VARIABLE
C     RT  ROOT OF BX+C=0
C     RT1  REAL ROOT ONE
C     RT2  REAL ROOT TWO
C     IM1  IMAGINARY ONE
C     IM2  IMAGINARY TWO
C     IND  RESULT INDICATOR       CODING SCHEME  0=NO ROOT
C                                                1=ONE ROOT
C                                                2=TWO ROOTS
C                                                3=COMPLEX
C
C*************************************************************
      REAL A,B,C,RT,RT1,RT2,IM1,IM2
      INTEGER IND




      IF (Condition)
         THEN statement
         ELSE statement
```

Figure 6. -- Interactions with a Programming Template.

Changes to program modules can be recorded without much difficulty. Updated program documents can be regenerated at any time. An impact analysis can be quickly and automatically performed during program maintenance. For example, when a change to an input or output variable is made, a scan on the stored information in the data dictionary can be activated, and all those modules or files affected can easily be identified.

6.2 Enforcement of Programming and Documentation Standards

ADD helps the program development project management to enforce the adopted programming and documentation standards and conventions. It provides a means to establish unified naming conventions for assigning variable names, and names for files, records, and program modules. It establishes the standards for program documentation and program listing. It provides a basis for establishing standard testing procedures.

6.3 Automated Program Document Generation

ADD can generate program design and implementation documents automatically. The program document updating problem is, therefore, reduced.

6.4 Program Validation

ADD provides an automated means for validating the program with its design. During the implementation and maintenance stages, any difference on the module interface conditions between the actual implementation and the design can be detected by checking the program listing with the design. The checking can be automated. The designer and programmer must resolve the conflict before the program is permitted to run. The preselected and stored test data along with the program specifications is another means to validate the program implementation.

7. CONCLUSIONS

ADD integrates the program design and program documentation activities. This approach not only helps in providing better documented programs, but it also enhances the program design. Program design and implementation documents can be automatically generated without adding additional burdens to program designer and programmer.

The programming and documentation standards can be better enforced. Structured program design, implementation, testing, and documentation can be easily established. The validity of the program can be better verified.

ADD depends on the availability of an automated data dictionary and an on-line program editor. Additional software is necessary to generate the necessary documents. Software is also needed to assist in program testing and to verify the validity of the implementation. Further research and development efforts are necessary to optimally implement the proposed concepts and tool. Detailed design and implementation of ADD are scheduled to be reported in our forthcoming papers.

REFERENCES

Boehm, B.W., McLean, R.K., and Urfrig, D.B., "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software", IEEE Transactions on Software Engineering, March 1975.

Brewer, G.D., "Documentation: An Overview and Design Strategy", Simulation and Games, Vol. 7, No. 3, 1976.

Comella, P.A., Computer Software Documentation, Goddard Space Flight Center, Greenbelt, Maryland, January 1973.

Denning, Peter, "Smart Editors", CACM, Vol. 24, No. 8, August 1981.

Dijkstra, G.W., "Craftsman or Scientist?", Proceedings of the ACM 1975 Annual Conference, San Francisco, California, April 17-18, 1975.

Feiler, P. and Medina-Mora, R., "An Incremental Programming Environment", Technical Report CMU-CS-80-126, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, April 1980.

GAO, Improvement Needed in Documenting Computer Systems, U.S. GAO Report No. B-115369, U.S. Government Printing Office, Washington, D.C., 1974.

Gey, F., Professional Levels of Computer Program Documentation, Computer Science and Applied Mathematics, Lawrence Berkeley Laboratory, Berkeley, California, June 1976.

Gass, S.I., Computer Model Documentation: A Review and An Approach, NBS Special Publication 500-39, U.S. Government Printing Office, Washington, D.C., 1979.

Gass, S.I., Hoffman, K.L., Jackson, R.H.F., Joel, L.S., and Saunders, P.B., "Documentatior for a Model: A Hierarchical Approach", Communications of the ACM, Vol. 24, No. 11, November 1981.

Henderson, P., and Snowdon, R.A., "Some Design Criteria for Program Development Tool", University of Newcastle, Great Britain, August 1973.

IBM, HIPO: Design Aid and Documentation Tool, IBM, Poughkeepsie, NY, April 1973.

Lefkowitc, H.C., "Data Dictionary Systems", Q.E.D. Information Sciences, Inc., Wellesley, MA, 1977.

NBS, Guidelines for Documentation of Computer Programs and Automated Data Systems, FIPS Publication 38, U.S. Government Printing Office, Washington, D.C., 1976.

Plagman, B.K., "Data Dictionary/Directory System: A Tool for Data Administration and Control", Auerback Information Management Services - Data Base Management, No. 22-01-02, Auerbach Publishers Inc., Pennsauken, NJ 1977.

Stevens, W.P., Myers, G.J., and Constantine, L.L., "Structured Design", IBM Systems Journal, Vol. 13, No. 2, 1974.

Teitelbaum, R., "The Cornell Program Synthesizer: A Tutorial Introduction", Technical Report TR-79-381, Computer Science Department, Cornell University, Ithaca, NY, July 1979.

Teitelbaum, R., and Reps, T., "The Cornell Program Synthesizer: A Syntax-directed Programming Environment", Technical Report TR-80-421, Computer Science Department, Cornell University, Ithaca, NY, May 1980.

Teitelman, W., "The INTERLISP Programming Environment", IEEE Computer, April 1981.

An Approach to Computer Maintained Software Documentation

Bruce I. Blum

The Johns Hopkins University
Baltimore, Maryland 21205

The use of text processors to manage documentation is quite common in data processing facilities. Consequently, much of the software documentation is produced in this manner. Unfortunately, we are not realizing the full potential of automation in the production of the documentation required for the different phases of the life cycle. This paper shows how one system is being designed to meet the documentation needs of the various users in a cost-effective way. The system was developed for a specific class of application - the moderate sized Information Management System (IMS). However, the approach is readily transportable to other application areas.

Keywords: Computer maintained documentation; Documentation requirements; Integrated design and documentation.

1. IMS DOCUMENTATION REQUIREMENTS

We define an IMS application as one which is data base oriented, uses off-the-shelf hardware and software systems, has no real-time demands beyond those associated with user interaction, and is limited to simple control logic which can be implemented as a closed algorithm or a decision table. A moderate sized IMS requires from 1 to 20 man-years of initial development and has a useful life of 5 or more years. Typical IMS applications have more than one class of user in an organization; most commerical and medical information systems are examples of this type of system.

Figure 1 illustrates the general processing flow associated with the development of an IMS. The Application Environment is the user's "real world." The implementation process begins with the formalization of the system requirements in the Descriptive Environment. The functions of the resultant descriptions are:

- To feed back to the user (sponsor) what the system is to do.
- To supply the detailed designer a set of specifications which he can translate to an implementable solution.
- To provide a foundation for the final system documentation required by the users, e.g., operations manual, introductory descriptions, etc.

As segments of the description are completed, they are transformed to unambiguous specifications in the Design Environment and finally an executable product in the Implementation Environment. The latter is run in the System Operations Environment. Once the system becomes operational, the design and implementation details are of little interest to the user, and hence their documentation is not shown. Life cycle support for a completed system is managed by a process called system maintenance. Since software does not wear out, system maintenance involves (1) correction of previously undetected errors and (2) expansion or modification of the operational system. The figure shows this as a feedback loop which must go through each of the development (and documentation) processes.

Using this model of the IMS life cycle, five users of the system documentation can be identified.

- Sponsor. Requires high level, descriptive materials which define the key design decisions and requirements in terms of the Application Environment.

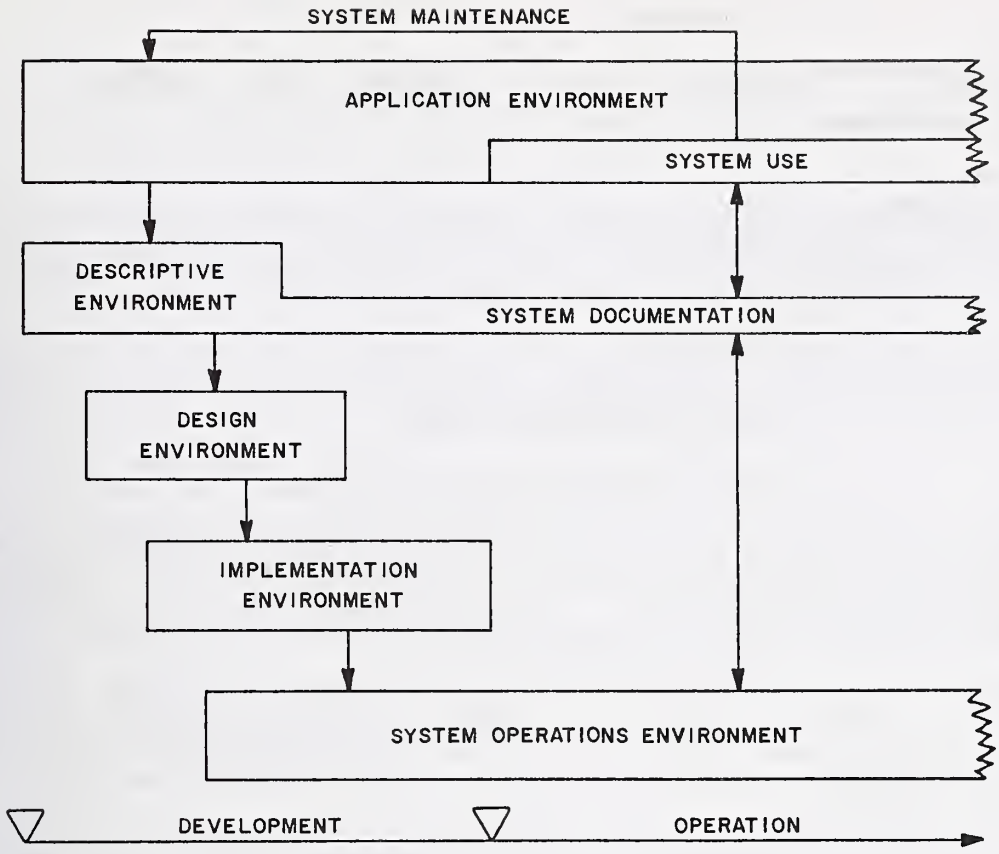


Figure 1
A Model of the System Life Cycle

- Designer. Requires materials which are both descriptive and highly specific. His documents communicate between designer and sponsor, designer and programmer, and among designers. They also provide a history and reference to be used by the author.

- Programmer. Requires highly detailed and specific information which will allow the translation of the input document into executable code. Note that if the design is directly translatable, this step and the need for a programmer (rather than designer) is eliminated.

- User. The needs of the user vary according to the user's involvement with the system. It may range from general introductory text to detailed scenarios.

- Operator. The operators' manual will vary in contents according to the system; it is always required.

Other specialized users of software documents are involved in functions such as system test and configuration control. For clarity, they are not considered here.

Figure 2 presents a matrix of documentation user by phase in the development life cycle. This paper presents a document development philosophy which meets the different and changing needs of each reader in a cost-effective manner.

AUDIENCE

System Phase	Sponsor	Designer	Programmer	User	Operator
Preliminary Design	X	X			
Design	X	X	X		
Code/Test		X	X		
Operational Use	X	X		X	X
Maintenance*	X	X	X	X	X

*Maintenance involves all of the above steps.

Figure 2
Document Requirements Matrix

2. THE TEDIUM APPROACH TO SYSTEM DEVELOPMENT

The Environment for Developing Information and Utility Machines (TEDIUM™) is a tool designed to implement and maintain IMS applications. The organization and philosophy of TEDIUM have been described elsewhere [1,2,3]. For the purposes of this paper it will be sufficient to familiarize the reader with some key characteristics of the system.

TEDIUM provides a comprehensive environment for the development of an IMS. It begins in the Descriptive Environment by supporting the designer in (1) the documentation of the requirements and (2) the decomposition of the requirements into a data-process flow using structured analysis techniques. An example of how this is done is illustrated with an appointment system example [2].

Once the initial design documentation is available, TEDIUM provides an application oriented specification language which fully defines the implemented units, i.e., the programs. Three types of specification are supported:

- Data base schema. TEDIUM uses a relational model with relations called "tables." Each table is decomposed into "index elements" and "data elements." All elements are variable length; multiple index elements are allowed. A special data type called text manages both text and format commands.

- General formatting rules. Every program is partially defined by a "frame" which defines the formatting rules to be used, e.g., is this an interactive device, what should the heading and foot lines for each page be, etc.

- Specific functional descriptions. Each program has its functionality uniquely defined by a program specification. Some specifications have been generalized (e.g., file management functions), others are described using the TEDIUM language.

TEDIUM generates executable programs from these specifications. Thus, there is no traditional programming in TEDIUM. Design and implementation involve the creation of readable specifications which establish the users' requirements and the detailed processes to satisfy them.

To illustrate how TEDIUM is used to implement an IMS, consider a hospital locator system. The first step is to describe the requirements. A single level outline of the requirements might be:

1. Enter Patients into the Locator
2. Query the Locator by Location
3. Query the Locator by Patient
4. Discharge a Patient
5. Transfer a Patient

6. Print a Midnight Census

After further analyses, this outline is modified and expanded. Requirements which might not be initially obvious are added (e.g., maintain a dictionary of hospital locations). Each item in the outline is then described in text, and a requirements statement is printed for sponsor review.

After these requirements are accepted, the designer translates the requirements (what is to be done) into a process-data flow design (how it is to be done). Definition of the data base and processes is done in parallel; there are no simple maps from the requirements to the process or data descriptions.

The definition of both processes and data groups continues by using the same outline approach. The process outline might combine requirements 1, 4 and 5 into a single process called Maintain Locator File. A single level outline of the processes might be:

1. User Menu
2. Maintain Locator File
3. Display Census for a Location
4. Display Patient Location
5. Print a Midnight Census

The single level outline of the data groups might be:

1. Locator File
2. Patient Index to Locator File
3. Patient Identification
4. Locator Identification

Again, in each case the outline is expanded and descriptive text is added. The audiences for this descriptive text are both the sponsor and the implementor/maintainor.

Figure 3 shows the three major classes of documentation text and the links among them.

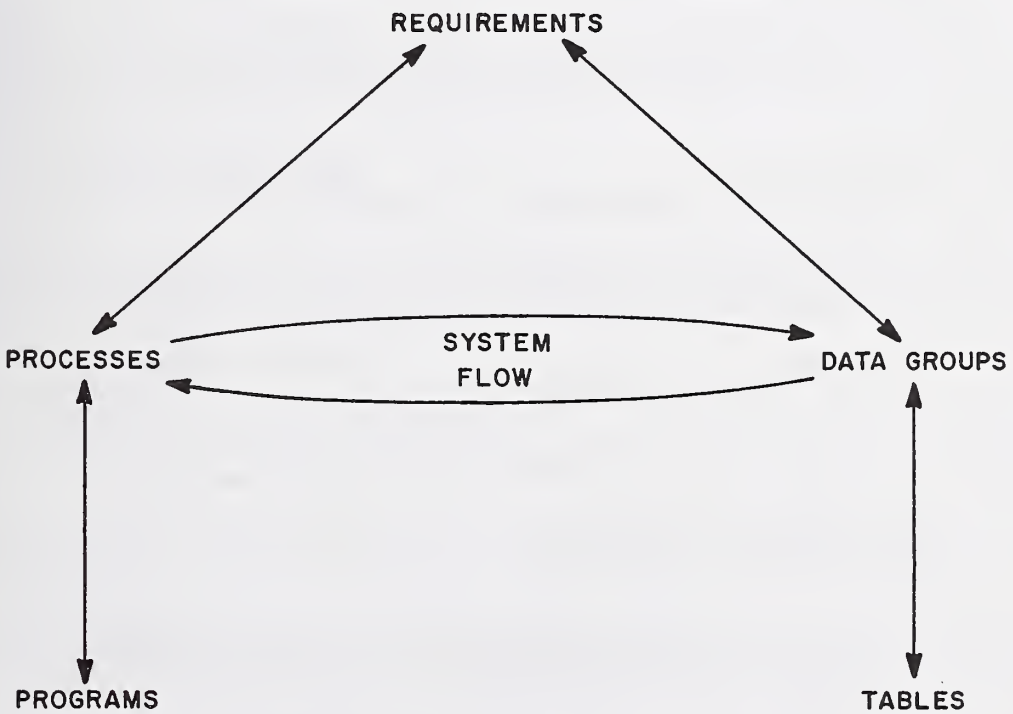


Figure 3
TEDIUM System Definition Tree

At the lowest level, the processes are linked to the programs which will be implemented; the data groups are linked to the tables (i.e., relations) which define the data model. Each program, in turn, is defined as a Frame (i.e., general set of formatting rules) and a minimal specification. The minimal specification is the least amount of information required to define what a program is to do. These specifications average 14 lines while the programs generated from them are 5 to 20 times longer [1]. The tables are defined in a schema which includes a data dictionary. Both the specification and schema entries have an initial block of descriptive text which describes their function or purpose.

Figure 4 contains a schema for the LOCATE table. In the short notation, this is

```
LOCATE     PATIENT LOCATOR


     This table serves as the inpatient locator. It is
organized by unit and contains only the patients who
are currently in the hospital.

  INDEX TERMS :
LOC        LOCATION                              CHARACTER  (5)

               This is the standard hospital unit identifier.

NAME       PATIENT NAME                          VARIABLE LENGTH  (31)

               This is the patient name in the form last,first
          truncated to the first 31 characters.

HNO        HISTORY NUMBER                        CHARACTER  (7)
           PATTERN 7N

               This is the standard 7 digit history number.


  DATA TERMS  :
ADMDAT     ADMISSION DATE                        DATE  (8)
           ROLE IN TABLE IS DEFAULTED TO TODAY

               This is the date that the patient was admitted
          to the hospital.

BED        BED NUMBER                            VARIABLE LENGTH  (5)
           ROLE IN TABLE IS MANDATORY

               This is the bed number within a hospital unit.

EDSDAT     ESITIMATED DISCHARGE DATE             DATE  (8)
           ROLE IN TABLE IS OPTIONAL

               This is the estimated date for patient discharge.
          It is normally modified each day during the last week
          of stay and is used for discharge planning and census
          projection.
```

Figure 4
Sample Table Schema

written

```
LOCATE(LOC,NAME,HNO)=ADMDAT,BED,EDSDAT
```

Since the data are stored in alphabetical index order, the table provides an index by patient name for each hospital location. The term HNO is included to provide uniqueness in the event of two patients with the same name in the same unit. (This is for illustration only, since a hospital will not assign patients with similar names to the same unit).

A minimal specification for the program to admit a patient (i.e., generate an entry in LOCATE) might be:

```
Input     LOC
Call      SETHNO
Input     ADMDAT
Input     BED
Input     EDSDAT
Put       entry in LOCATE
```

Input is a command which prompts for the element using its dictionary name (e.g., "LOCATION" for the element LOC), checks the input for validity and stores the correct data. It also accepts the "?" as a help prompt. The designer may supply a help message; the default is a printing of the data dictionary definition as shown in Figure 4. The program SETHNO prompts with ENTER NAME OR NUMBER, processes the input, adds new patients to the patient identification tables, and always returns with NAME and HNO set. Put writes the completed entry into the data base.

The process of design and implementation with TEDIUM, therefore, involves the description of the problem in a structured text outline followed by the definition of the data model and programs with the TEDIUM specification tools. Programs are generated from these specifications and tested in a rapid prototype environment [1]. All text, specifications and linking information are available online.

3. THE TEDIUM APPROACH TO DOCUMENTATION

TEDIUM is a set of programs generated by TEDIUM. Thus, the preparation of documentation for an application is seen as an IMS application. All requirements, processes and data group text is stored as tables managed by the TEDIUM programs. To illustrate, the requirements are stored in two tables:

```
REQ(REQID)=REQNM,REQTX
REQL(REQID,REONO)=LREQID
```

REQ is the table which defines the requirement text. The index (REQID) is a short, variable length identifier. REQNM is the requirement name and REQTX is the text description. (The text data type calls the TEDIUM text processor for all text input and editing functions.) For example, we might use

```
REQID="ENTER"
REQNM="ENTER PATIENTS INTO THE LOCATOR"
REQTX="The system must enter patients into its locator file.  By this it is
       meant that ...."
```

Note that the text avoids phrases such as "the system shall." This allows use of the requirements text in a wide range of documents throughout the life cycle.

The second table, REQL, establishes a tree structure for the requirements. REQNO is a number which sequences this node among those nodes directly linked to REQID. LREQID contains the value of a REQID in REQ. For example, the following entries in REQL

```
REQL("ENTER",10)="ENTER.NEW"
REQL("ENTER",20)="ENTER.OLD"
```

conform to the outline

```
ENTER PATIENTS INTO THE LOCATOR
     NEW PATIENTS
     RETURNING PATIENTS
```

The links between text groups are described in table pairs to provide bidirectional linkage. For example, to link requirements and processes we use

```
REQPRO(REQID,PROID)
PROREQ(PROID,REQID)
```

Links to the schema and specifications provide access to their text documentation. These are defined in

```
TID(TID)=TNAME,TTX
SID(SID)=SNAME,STX
FID(FID)=FNAME,FTX
```

for tables, specifications and frames respectively. The element NAME is a 30 character short name, and TX is a block of descriptive text which describes the functionality or use of the item identified. Finally, within a specification there are links to the optional help text (i.e., response to a "?" input).

All documentation of a TEDIUM application is created from this data base. To manage this, one additional set of tables is defined for requirements, processes and data groups. For requirements it is

```
REQT(REQID,REQUS)=REQLV,REQFM
```

where REQUS is the user who would be interested in this block of text; REQLV is a level of importance with 1 being the greatest and 10 indicating details of little general interest; and REQFM contains formatting instructions such as eject a page, do not print the heading, follow with the full tree for this node, etc.

With this table, one can now create documents. For example, a system overview might be:

```
Print all requirements with REQUS="general" and REQLV less than 3.
```

A Users' Manual for the patient locator function as one application in a total hospital information system might be:

```
Print a title page with "Users' Manual, Patient Locator Functions"

Print "Introduction" as all requirements with REQUS="general" and REQLV=1

Print "The Patient Locator" as all requirements in the "LOCATE" tree with
     REQUS="general" or "user" and REQLV between 2 and 5

Print "General Instructions" as all processes in the tree "GEN" (This is a
     general set of instructions for the use of all TEDIUM generated systems.)

Print "Locator Flow" as processes in the tree "LOCATE" with PROUS="general"
     or "user."

Print "Data Dictionary" and print outline of "LOCATE" data groups with linked
     tables.  Follow with an appendix containing all the table schema.
```

The same document may be described more briefly as: User Manual for "Patient Locator Functions" using requirement node "LOCATE," process node "LOCATE" and data group "LOCATE."

A final level of detail can be produced for the users' manual by processing the specification and its help messages. For example, the program to enter a patient might be documented by:

```
(Lead in text from the process tree)
"The system will now prompt:
     LOCATION"
(Help message text)
"The system next prompts
     ENTER NAME OR NUMBER"
(Help message text from program GETHNO)
     :
     :
```

Each of the above documents is directed to users and designers. The availability of all text and specifications in a data base also provides an information resource for those responsible for the implementation and maintenance of the application. TEDIUM supports online (and printed) access to the following:

- Text data by node or tree
- Table definition and the data dictionary
- Specifications for programs and frames
- Cross-reference of data element by table and use (index/data)
- Cross-reference of programs (calls/called by)
- Cross-reference of programs with tables and vice versa (reads/writes)
- Hierarchical relationships among programs (a tree structure showing how programs are called.)

Since all of these outputs are generated directly from the data base, they always accurately reflect the current application design.

4. CONCLUSION

This paper has briefly (1) presented a model which describes the kinds of documentation required for an information management system, (2) shown how the text and specifications for such an application can be managed in the design environment, and (3) illustrated how the resultant design data base can create the necessary formal system documentation.

Two types of documentation were identified: (1) material to be distributed to sponsors and users and (2) design information required for implementation and maintenance. The former is always printed; the latter is always available online. The computer stored data base is the primary source and reflects the most current documentation. Given the proper discipline for configuration control, changes to the system will be entered into the documentation, and updated documents can be created with minimal effort. Since the lower level, implementation oriented documentation is derived directly from the specifications, it will always accurately reflect the current design.

The system discussed (TEDIUM) has been in operational use for about a year in a MUMPS prototype. The main thrust in its use has been rapid prototyping and system development. It provides the work environment for a group of ten programmers. Several major systems which have been developed or converted with TEDIUM are now in operational use [4,5]. The documentation for these systems is being managed as described above.

Still, some disclaimers are in order. TEDIUM is not yet available for export. The approach to documentation is in a trial stage, and we are learning. Only a limited class of applications is being considered, and the problems of multiple versions and releases have not been addressed. Nevertheless, we are convinced that the concepts presented offer an integrated, workable methodology for inexpensively eliminating that ever present problem: missing or inaccurate documentation.

5. REFERENCES

1. Blum, Bruce I. A Tool for Developing Information Systems. (Eds.), H. J. Schneider and A. I. Wasserman, Automated Tools for Information System Design and Development, 1982, North-Holland Publishing Company.

2. Blum, Bruce I. and C. W. Brunn. Implementing an Appointment System with TEDIUM. Fifth Annual Symposium on Computer Applications in Medical Care, Washington, D.C., November 1-4, 1981.

3. Blum, Bruce I. Program Generation with TEDIUM, An Illustration. Trends and Applications 1981, National Bureau of Standards, Gaithersburg, Maryland, May 28, 1981.

4. Blum, Bruce I. and R. E. Lenhard, Jr. An Oncology Clinical Information System, Computer Magazine, November 1979, p. 42-50.

5. McColligan, Elizabeth E., Bruce I. Blum and C. W. Brunn. An Automated Core Medical Record System for Ambulatory Care, 1981 SAMS/SCM Joint Conference, Washington, D.C., October 30 - November 1, 1981.

Automated and Automatic Documentation

Linda K. Lawrie

U. S. Army Construction Engineering Research Laboratory
Champaign, Illinois

Software documentation standards should reflect the evolution of software engineering. Moreover, future progress in software development should be expected and allowance for progress should be incorporated in documentation guidelines.

Systems such as the Automated Documentation System may serve as prototypes for producing automated documentation. Both increased productivity and cost savings can be expected from requiring meaningful documentation.

Keywords: automated documentation; documentation standards; internal documentation; software engineering;

1. INTRODUCTION

During a recent review of documentation standards ([1], [2], [6]), two deficiencies seemed apparent: 1) the documentation requirements have little relevancy to scientific and engineering software development efforts; 2) software engineering has evolved beyond scope of current standards. These two deficiencies result in loss of programmer productivity, in increased code rewriting -- in general, in decreased cost benefits of the software. This paper will examine these two deficiencies and propose possible solutions.

2. SUMMARY RECOMMENDATIONS

This paper recommends creation of 'flexible' documentation standards. Because of the rapid evolution of software engineering, current procedures and practices may have an effective lifetime of less than five years. Improved productivity can result when the programmer views the documentation standard as effective and useful.

This paper further recommends automation of the software documentation, whether from well commented code or from text editing processors. This machine-readable documentation should form part of the complete software package. The preceding actions can result in increased productivity by allowing the programmer to document and update on the same hardware. If the major portion of the documentation is contained within the source code, then many kinds of documentation reports may be automatically generated.

In general, substantial cost savings -- particularly labor costs -- can result from adoption of these recommendations: programmer productivity increases may result from improved motivation; effective software life may be extended.

3. DEFINING THE PROBLEM

1) The documentation requirements have little relevancy to the actual workings of engineering software. It seems obvious that documentation should aid the understanding of the software and how the software operates. Most documentation standards seem to require only superficial details -- file names, record layouts, etc. ([1], [2], [6]) These requirements may be applicable to typical batch oriented COBOL environments used in most financial and business applications, but do not apply to some of the Fortran environments used in scientific and engineering applications. Further, the documentation standards do not easily lend themselves to detailed explanations of complicated software.

What are the 'record layouts' in a system where graphic input (e.g. joystick, digitizer, light pen) are used? Likewise, how does one describe 'record layouts' for English-like language interfaces processed using formal grammars? Even though these kinds of systems may not presently form a majority of the software developments, these kinds of systems (interactive, graphic input, user-friendly interfaces) are the trend of the immediate future.

When trying to document one of these systems according to the standards, programmers become discouraged when the specifications don't appear to apply. Perhaps some recognition of these kinds of systems (e.g. interactive, flexible input requirements, etc.) would aid usage of the documentation standards. Also, recognizing that software requirements do evolve would enhance documentation standards.

2) Software engineering has evolved beyond the scope of the standards. This statement is true regardless of the language -- COBOL or Fortran. Current software is being developed and used on multi-programming, often interactive hardware. User-friendly interfaces are being required. Thus, the standard documentation requirements of frequency of runs seem useless. Frequency of runs -- anytime the user wishes -- are not as important as when these frequencies were required for computer resource scheduling.

Superficial knowledge of the software may be sufficient after the software is placed into production, but the software development manager, as well as the software maintenance programmer, needs far more detailed knowledge about the actual code. Comments placed into well written code (internal documentation) will aid the maintenance programmer as well as a development manager. To enhance productivity, it would be nice if these well commented routines could be used to satisfy post-implementation documentation requirements.

As the evolution of software engineering progresses, the manager and user are becoming more cost-accountable (i.e. like other departments). Software development is being scrutinized for cost benefit, timeliness, and effective life. Users, likewise, are becoming more sophisticated both in specifying reasonable desires and in recognizing usable software products. Thus, more importance is being attached to management and user knowledge of developing software; managers and users also need to know what the code does -- though not necessarily at a detailed level.

Internal documentation -- well commented code -- can solve most of these requirements; external documents need only be used for overviews and more detailed descriptions. If internal documentation can be used to produce readable documents, comments will be considered almost automatic documentation.

Originally, scientific processors were merely number crunchers; text processing -- user manuals, documentation, etc. -- was written/dictated by programmers and then typed. Typing has since turned into word processing, typically on different hardware than used for software development. This divergent hardware seems less necessary now that most scientific processors have good text editors and text processing programs.

Rather than the redundancy of having software documentation on separate hardware, it would be cost beneficial to have the programmer produce the documentation on the development hardware -- ideally, internal to the code itself.

4. POSSIBLE SOLUTIONS

4.1 Usable and Automated Documentation

Any possible solutions must emphasize 'usable' documentation. These solutions must also be cost beneficial -- or they won't be enforced. The documentation audience -- manager, user, and maintanence programmer -- should be satisfied with the result. Many problems can be avoided by the use of structured programming enhanced with mnemonic variable name usage. Structured programming will help produce good code because of its emphasis on modular design. Guidance has begun to appear for using structured programming [3].

Automated production of documentation reports will also help, particularly if these reports can be produced directly from the source code(s). Then, new reports can easily be produced whenever changes occur in the code. Also, the documentation will then be wherever the source code is found. This automation seems most easily done by commenting code; external documentation can focus on less changeable items of the software -- purpose, overall techniques.

4.2 The Automated Documentation System

Rather than theorizing about such a system, I can tell you about one that already exists. The Automated Documentation System (ADS) [4] was developed at the Construction Engineering Research Laboratory (CERL) to provide high quality documentation of Fortran source code(s).

The Automated Documentation System (ADS) is a computer program and user procedure designed to facilitate management of the development of software and the production of final documentation for Fortran programs. The ADS system can be used in two ways. 1) At any point during development of the software, the status of the development process can be determined by the application of the ADS program to the source code under development. Flow charts and internal documentation are summarized for the project manager. 2) After the software is complete, external documentation can be produced from the internal documentation by running the ADS program.

The ADS program is written in Control Data Corporation (CDC) Fortran extended, and can be used on CDC 6000/7000/170 series computers with few or no modifications. The source code is non-proprietary.

The Automated Documentation System (ADS) can be viewed as a prototype of the necessary systems for obtaining good documentation. Similar methods could be used for COBOL, Pascal, ADA, and other languages. ADS relies on special comments placed <u>into</u> the Fortran source, stores these comments along with other, compiler provided information on software data files, and retrieves and reformats this information from these files for specialized reports. The ADS user -- programmer -- can choose from 20 different sections for documenting his code (e.g. PURPOSE, METHOD, VARIABLE DICTIONARY, FLOW, etc.). Compiler supplied information includes number of lines of code, routines used by a routine, variables and their types, etc.

The software project manager can specify which sections are to be used in a developing software system. When top down design is employed, a system similar to the ADS Stub Generator [5] can be used to enforce the manager's requirement. However, meaningful documentation -- the documentation accurately representing the purpose of the software -- implies a careful review process. Thus, the reviewer (manager or user) needs to check the documentation for meeting the 'standards' requirements <u>and</u> to review the documentation for content.

An example application of ADS to the ADS main program can be viewed in Figure 1.

```
LOGICAL FLOW.
      CALL INITIALIZATION ROUTINES (ADINIT, FHINIT, PINIT)
      WHILE NOT EOF(REFFL) DO [SOURCE CODE PROCESSING]
           PROCESS REFERENCE MAP FOR A ROUTINE
           SCAN SOURCE CODE FOR CURRENT ROUTINE
           PROCESS DOCUMENTATION FOR CURRENT ROUTINE
      END [SOURCE CODE PROCESSING]
      PERFORM USER INPUT COMMANDS [TITLE, PRINT, DRAW, ETC.]
      PERFORM END-OF-JOB PROCESSING

           FILES USED IN ADSDOC  ARE --
      DEBUG     INPUT     INVBK     INVFL     MASBK     MASFL
      OUTPUT    REFFL     RPTOUT    SRCFL     TIMES

      THIS MAIN PROGRAM HAS APPROXIMATELY     112 NON-COMMENT LINES.

      IT WAS WRITTEN IN FORTRAN 66.
```

Figure 1. Excerpt of ADS documentation.

ADS Reports can be retrieved for any combination of sections, routines, and common blocks. These reports can be read by anyone, if the comments are meaningful; no one will have to sift through the Fortran code to understand the software. These reports can be produced at any point in the software development and can provide the manager and user with crucial information about development progress. Several kinds of reports currently exist in the system -- reports of documented ADS sections for routines, reports of documented ADS sections for common blocks, and a 'tree report' -- static calling structure from any specified 'root' module.

4.3 ADS Examples

Though space limitations do not allow complete examples of ADS outputs to be shown, some excerpts from ADS inputs and outputs may be useful. The retrieval user communicates with the ADS program using simple English-like commands:

```
TITLE SOFTWARE IN PROGRESS;
REPORT IGUIDE=TITLE,PURPOSE,USAGE,FORMAL PARAMETERS;
PRINT NARROW (IGUIDE) FOR ALL ROUTINES;
DRAW TREE;
END;
```

The Automated Documentation System program places information on the documentation files from the TITLE and REPORT command. It retrieves the necessary documentation records for the PRINT command; and it draws the static calling structure of all main programs in this example DRAW command.

Again, the programmer places special comment cards into the Fortran source code. These comment cards, together with the information provided by the Fortran compiler reference map, create the documentation record. For an example of ADS output produced by the compiler reference maps, refer to Figure 2.

```
     ROUTINES CALLED BY AVGR    ARE --
TRACER

     INTRINSIC FUNCTIONS CALLED BY AVGR    ARE --
IABS

     COMMON BLOCKS CALLED BY AVGR    ARE --
DEBUGR    SUMCOM    COMMAR    COMMDEF

     THE ROUTINES WHICH CALL AVGR     ARE --
PRNTSUM
```

Figure 2. Fortran Reference Map produced documentation.

For an example of typical ADS output produced from the special comment cards, refer to Figure 3. Tree reports may be 'drawn' starting at any module and continuing for a user specified depth. An example of the two top levels of ADS itself can be viewed in Figure 4.

```
TITLE.
AVGR - AVERAGING ROUTINE

PURPOSE.
AVGR CALCULATES AN AVERAGE AT THE REQUESTED REPORT FREQUENCY (INTERMITTANT AVERAGE)
OR A GRAND AVERAGE AT THE REQUESTED REPORT FREQUENCY+1 (I.E., DAILY FOR AN HOURLY
REPORT) BASED ON THE INPUT CODE, AND ZEROES OUT THE APPROPRIATE INTERNAL SUM BUFFER.

USAGE.
CALL AVGR(CODE,VARTYP,BUFFER,LTH)
ENVIRONMENT --
  THE INTERNAL FLAGS DESIGNATING WHICH ARITHMETIC OPERATIONS ARE ALLOWED ON EACH
VARIABLE MUST BE SET PRIOR TO A CALL TO AVGR.
  IF THE REPORT WRITER LANGUAGE AND PARSER ARE USED ALL OPERATIONS ARE ALLOWED ON
ALL VARIABLES INPUT.  THE VARIABLES AND THEIR ALLOWABLE OPERATIONS MAY
ALTERNATELY BE SET WITH A CALL TO INITV.
  THE INPUT DATA BUFFER MUST ALSO BE FILLED WITH HOURLY DATA AND/OR 'SPECIAL' OR
'NODATA' FLAGS FOR EACH VARIABLE REQUESTED FOR THE CURRENT REPORT BEFORE EACH CALL.
THIS IS ACCOMPLISHED BY A CALL TO 'SETID' FOR EACH USER-SPECIFIED ZONE, SYSTEM OR
PLANT NUMBER (ID) IN THE REPORT THAT REQUESTS RPTFLE DATA, AND BY A CALL TO
'RDFILES' EACH HOUR TO RETRIEVE THE DATA FOR ALL REQUESTED VARIABLES.

VARIABLE DICTIONARY FOR ROUTINE AVGR

FORMAL PARAMETERS.

     BUFFER  - 1-DIMENSIONAL ARRAY IN WHICH RESULTS OF AVERAGE ARE RETURNED.

     CODE    - ACTION CODE FOR AVERAGING (NEGATIVE CODES ALSO ZERO BUFFER)
               1= PERFORM AN INTERMITTANT AVERAGE, STORE RESULT IN INTERNAL BUFFER
               2= PERFORM A GRAND AVERAGE, STORE RESULT IN INTERNAL BUFFER
               -1= RETURN CURRENT INTERMITTANT AVERAGE FROM INTERNAL BUFFER
               -2= RETURN CURRENT GRAND AVERAGE FROM INTERNAL BUFFER

     LTH     - LENGTH OF BUFFER

     VARTYP  - RECORD TYPE OF DATA TO BE AVERAGED
               (1=ZONE, 2=SYSTEM, 3=PLANT, 4=SYSTEM ZONE)
```

Figure 3. Comment produced documentation

To ensure meaningful documentation, these reports are used in structured walk-thrus of code and in meetings with users and managers. Reviewing this documentation for compliance with development standards and for completeness of meaning becomes the responsibility of the managers and users. Using these reports also eliminates the tendency of programmers to say 'the code is self-documenting'. Of course, these reports cannot substitute for code, but they do provide a common ground for discussions, changes, etc. When implementing changes, the programmer need only update the routines/common blocks changed, and rerun ADS; software documentation files will be updated to reflect the changes.

```
ADSDOC__
        |_ADINIT
        |_DTFTN___
        |         |_RDCDC4
        |         |_RDCDC5
        |         |_RDLINE
        |_FHEND
        |_FHINIT
        |_PINIT
        |_PREPARS_
        |         |_CDPARS
        |_RPTMON__
                  |_DELCMD
                  |_DRACMD
                  |_HELCMD
                  |_PRTCMD
                  |_RPTCMD
                  |_TITCMD
```

Figure 4. A TREE report showing user specified depth.

Certain documentation reports may not easily fit into the 'internal' comment structure. These kinds of reports usually deal with overall views of the software or intricate interactions of several routines, programs, etc. For these cases, one can still produce the document in machine-readable form using a text processor and carry these documents as part of the software. Thus, all the available documentation about the software would be available as one package.

4.4 Automated Documentation System Usage

The Construction Engineering Research Laboratory developed the Automated Documentation System to meet documentation needs of users. ADS has been used in several systems produced at the Construction Engineering Research Laboratory since December 1978. Retrofit documentation of several large systems has also been accomplished using the special comments required by ADS. In particular, the Building Loads Analysis and System Thermodynamics (BLAST) program -- approximately 100,000 lines of Fortran source code -- that has been installed on 40 different sites uses the ADS commenting system. Producing the documentation for BLAST, then, is merely a matter of executing the ADS program against the BLAST documentation files. In the case of the BLAST system, ADS is being used primarily in program maintenance.

However, several other users of ADS at CERL have been using the ADS system primarily as a system development tool. These users have found the automated reports will help them identify mis-typed variable names, as well as provide indications of module completeness.

4.5 Status of the Automated Documentation System (ADS)

The ADS system periodically undergoes some revision primarily to respond to new needs of its users. The system is somewhat prototype (though used productively) since the code exists primarily for Fortran and on Control Data Corporation hardware.

Moving ADS to new hardware requires some code revisions and creating reference map decoding routines for the new hardware's Fortran compiler. Obviously, a portable Fortran code scanner that produces the kind of information used by ADS (e.g. externals, intrinsic functions, variable-common mapping) would aid conversion to new hardware.

Using ADS for languages other than Fortran would be more difficult. For example, many of the terms used in ADS are not applicable to COBOL (e.g. lack of common blocks, subroutines not widely used). However, the basic concept of ADS -- storing documentation on files for later retrieval -- could be applied to many source languages.

5. CONCLUSION

Documentation standards must be applied to a wide variety of software systems. Originally, software was used in batch oriented, single programming environments. However, present and future software will be used in interactive, multi-programming environments. Thus, any documentation standard must be flexible in its requirements.

The need for meaningful documentation exists throughout the software life cycle. Software development managers need to be able to easily assess the progress of development. Users need to be able to assure themselves that they are getting the product that meets their needs or desires. Maintanence programmers need to have system documentation readily available and easily revisable.

The Automated Documentation System (ADS) system was developed to meet these needs at a specific installation (the Construction Engineering Research Laboratory -- CERL). Tools like ADS may also meet the needs of the documentation standards community.

6. REFERENCES

[1] DoD Standard 79365.1-S, Automated Data Systems Documentation Standards, 13 September 1977.

[2] TB 18-111, Army Automation Technical Documentation, January 1979.

[3] TB 18-103, Army Automation Software Design and Development, January 1980.

[4] Lawrie, Linda, The Automated Documentation System - User Manual, Cerl Technical Report E-147, February 1979, 83 p.

[5] Lawrie, Linda and Baugh, Jean, Automated Documentation System (ADS) Stub Generator: Description and User Instructions, Cerl Technical Report E-167, October 1980, 35p.

[6] FIPS Pub 38, U. S. Department of Commerce / National Bureau of Standards, 1976 February 15.

SESSION C: Discussion

H. Hecht

SoHaR, Incorporated

Currently available tools permit partial automation of document generation. Thereby, they reduce cost and schedule, provide standardization of format, and improve the quality of the resulting product. Tools can help in three areas of document preparation: collecting information, establishing the format, and in presentation to the user. The authors of this session covered all three areas as is shown in the following table.

DOCUMENTATION AREAS SERVED BY TOOLS

AREA	EXAMPLES	AUTHOR
Information Content	Variables dictionary, program flow, etc.	Lawrie
	Copies of actual program use	Henry
Format	Pre-arranged sequence of topics	Ting Lawrie
	Mandatory declarations	Malhotra Blum
Presentation	Screen format	Ting
	Text editing and formatting	Henry

The authors recognized limitations of current use and the methodology associated with their use, including:

* Difficulties in devising tools for general purpose documentation. The most successful tools were dedicated to a single application area.

* The full benefit of tools can be realized only if there is strong management support and follow-up.

* Current methods address primarily user and maintenance manuals.

Looking into the future, one may expect greater emphasis on interactive documentation. Menu-driven input, supplemented by HELP routines, is already widely used and reduces the dependence on user manuals. Screen presentation of fully documented source code, possibly structured at several levels of detail, may supplant much of the material currently included in maintenance manuals. Techniques that permit simultaneous interactive display of programs and associated data structures promise to be particularly helpful for program development and maintenance.

A hopeful sign is that the academic community may be taking some interest in documentation as evidenced by Ting's paper. That paper recognizes that there is a close relation between good program design readily understood documentation. This emphasizes that documents prepared for human understanding and programs prepared for use in the computer are complementary representations of the same information processing activities. Deficiencies in either one will usually degrade the other.

Session Summary: Tools for Improved Documentation

Sheila E. Frankel

National Bureau of Standards
Gaithersburg, Maryland

This session proposed the use of software tools in documentation development as a promising solution to some of the problems plaguing traditional program documentation—expense, limited usefulness of the final product, and the problem of keeping the documentation current as the project changes. It was agreed that part of the problem is that documentation is developed and maintained separately from the program. This is the case whether it is written in parallel with program development or whether it is done as a follow-up effort.

Linda Lawrie suggested another problem—that documentation standards and methods have remained static, while the technology of program development, or software engineering, is constantly changing and evolving.

One approach to help mitigate these problems is the use of automated documentation tools. These tools can make the development of documentation part and parcel of the programming development process. Several different approaches to the use of tools have been suggested.

Ashok Malhotra's approach is the development of a high-level programming language that is easy to understand and, thus, self-documenting. In the EAS-E system, by integrating this language with a DBMS, a new entity has been created—executable documentation.

T. C. Ting's ADD system integrates program design and documentation. It imposes order on the design documentation through the use of a template. This information, together with pseudo-code and test data, is used to create structure charts and flow diagrams, which in turn can be used in the program development and validation.

Bruce Blum has suggested reversing the customary process of developing Information Management Systems. His TEDIUM system allows the developer to specify system requirements in terms of format, function and processes, and from these executable programs are generated.

Linda Lawrie follows the customary procedure of program development, but, in her ADS system, the documentation is either contained within the program or automatically generated through program analysis.

Ray Houghton presented the idea of including different types and levels of on-line documentation in an interactive program. He used as an example a system developed by Nathan Relles and Lynne Price that was demonstrated at a recent tool fair co-sponsored by NBS (See NBS Special Publication 500-80). This system enables the user to specify precisely the information and the level of detail he needs, and not to be distracted by receiving too much or too little information in response.

Lee Henry suggested that, once the documentation effort is integrated into the software development process, not only do we need standards for the contents of documentation, but for the processes and procedures of documentation development as well. She also suggested, as an addition to the program development team, a documentation specialist, who would be involved in all aspects of the software design.

These systems represent differing views of automated documentation—programs as documentation, programs generating documentation, documentation generating programs, and executable documentation as part of a program. In conjunction with all of these approaches, those documents that are not an integral part of the program can be produced using text editors and word processors. Herb Hecht summed up by stating that surveys have shown that documentation tools are well-liked by programmers.

However, two reservations were expressed with regard to this promising approach to documentation. Herb Hecht commented that fully automated documentation is not yet feasible. Human intervention is still needed to disclose general intent and patterns in the programs. In addition, all of the participants admitted that their systems had limitations. All of these systems are either prototypes, are available only on specific machines, or are suitable only to specific types of systems or applications. Frequently, tools are incompatible with other tools available on the same system.

A concern expressed by several members of the audience was whether tool usage to aid or replace the programmer would further increase the distance between software developers and users. Several of the panelists responded that the major part of a programmer's time is currently spent on "housekeeping tasks." The tools can take over these tasks—by removing them from the programmer's domain (Blum), hiding them from the user (Ting), or eliminating the distinction between specifications and programs (Malhotra). This allows the programmer to concentrate on the functional part of the program, and on identifying and satisfying the users' needs.

An additional theme was repeated a number of times. In order to capitalize on any of these advances in documentation development, management must become committed to the necessity of documentation as an inseparable part of the software development process.

This session demonstrated that, through the use of automated tools, documentation development can be a dynamic process, reaping all the benefits of the new advances in technology.

SESSION D: Do Existing Standards Work?

Alfred R. Sorkowitz

U.S. Department of Housing and Urban Development

Introduction

ADP systems are recognized as evolving from initial concept to final operation. This evolution is called the life cycle and is characterized by several major events or milestones which delineate the life cycle phases of the system. These milestones provide the means of measuring, evaluating, and thereby controlling the progress of the development effort. It is recognized that there are many different terminologies to identify the phases and the associated documentation.

In February 1976, the National Bureau of Standards issued the Federal Information Processing Standards Publication (FIPS PUB 38): "Guidelines for Documentation of Computer Programs and Automated Data Systems" (February 15, 1976). This document presents detailed content guidelines for ten document types generally prepared during the Development Phase.

This was followed in 1979 with FIPS PUB 64 "Guidelines for Documentation of Computer Programs and Automated Data System for the Initiation Phase (August 1, 1979).

Taken together, these two FIPS PUBS are the first attempt at a government-wide Documentation Standard and today are widely used within the Federal ADP Community.

a. Initiation Phase

During the Initiation Phase, the objective and general definitions of the requirements are established. Feasibility studies, cost benefit analyses and other documents, determined by agency procedures and practices, are developed.

b. Development Phase

The Development Phase is broken down into four stages; Definition, Design, Programming, and Test.

The DEFINITION stage defines the functional and performance requirements needed to initiate the design as well as to confirm that the completed software will meet the objectives. These functional Requirements define what the software must do rather than how it is to be done.

During the DESIGN stage, the System and Program Specifications are developed. These are detailed specifications of the internal construction of the software, for use by programmers who will implement the design.

During the PROGRAMMING stage, program language statements or "code" is produced that meets the Design Specifications. Program or Unit Testing also takes place during this stage.

During the TEST stage, System, Integration and Acceptance Testing takes place according to a Test Plan prepared in a previous stage.

c. Operation Phase

During the Operation phase, software is maintained and enhanced as additional requirements are identified.

Sufficient experience in the use of these documentation standards now exists, and it is appropriate to stop and evaluate the present standards. Therefore, the title of this session "Do Existing Standards Work"? The focus is on how these standards can be modified to be responsive to new initiatives as well as new methods and procedures for preparing the required documents.

Ronald Thies in his paper "Documenting System Security" addresses the new emphasis on System Security as a result of OMB Circular A-71 TM No. 1, "Security of Federal Automated Information Systems." The paper presents an approach to documenting system security which provides for the threading of security throughout the documentation life cycle. The approach requires that security requirements are thoroughly described in the Functional Requirements Document and that security is specifically addressed as it applies to each of the subsequent documents.

It is interesting to note that these changes have successfully been accomplished within the FIPS PUB 38 framework.

Traditionally, documentation is prepared by the developers. Personnel working on System Requirements prepare the Requirements documentation, System Designers prepare the Design Documentation, etc. Patrick O'Conor and Samuel Redwine described an environment where for unique reasons this method didn't work. In their experience in establishing a technical writing organization for the purpose of preparing ADP software documentation.

The final paper in this session by Robert Hegland is entitled "An Overview of the Department of Defense Automated Data Systems Documentation Standard - an adaptable standard." It briefly describes the DOD standard, which is very similar to FIPS PUB 38 and in fact was used as a point of departure by the intergovernmental team that developed the NBS Standards.

Documenting Systems Security

Ronald G. Thies

Systems Specialist, Standards and
Quality Control Staff
Office of ADP Systems Development
U.S. Department of Housing and Urban Development (HUD)

This paper presents an approach to documenting system security which provides for the threading of security throughout the documentation life cycle. The approach requires that security requirements are thoroughly described in the Functional Requirements Document and that security is specifically addressed as it applies to each subsequent document. Its structure provides for the convenience of security reviews necessary to attain system certification. To describe the approach, revised security sections were prepared for the Functional Requirements Document, System/Subsystem Specification, Test Plan, and Test Analysis Report which are described in the current FIPS PUB 38 (1).

1. INTRODUCTION

1.1 Background

With the issuance of OMB Circular A-71, Transmittal Memorandum No. 1, Security of Federal Automated Information Systems, on July 27, 1978, and subsequent guidance from NBS, OPM, and GSA (3) (4) (5) (6), documentating system security is being viewed with a new perspective in Federal Government. Operating agencies must now consider documenting system security an integral part of the development process not only for the very sensitive but for all systems. The old assumption that "a system is non-sensitive unless specifically designated sensitive" must now be replaced with the assumption that "a system is sensitive unless specifically designated non-sensitive." In other words system security must be addressed and documented at the outset of the development process to determine: (a) the degree of sensitivity to place on the system and (b) the amount of effort to devote to the development and documentation of appropriate security safeguards.

The documents in the current version of FIPS PUB 38 (1) address security in somewhat general terms. Agencies using this publication as their primary documentation standards will have to augment its coverage of security to assure that specific security requirements are adequately described in the Functional and Data Requirements Documents and that appropriate safeguards are described and evaluated in the subsequent documents. It is important that improved coverage of system security be included in the forthcoming revision to FIPS PUB 38 (1). This improved coverage should be structured so that it provides for the security reviews necessary to attain system certification, a specific requirement of OMB Circular A-71, Transmittal Memorandum No. 1 (2).

1.2 Purpose

The purpose of this paper is to present an approach to threading security throughout the documentation life cycle. It is hoped that this approach or a similar approach will be considered in the next revision of FIPS PUB 38 (1).

1.3 Scope

This paper contains proposed security sections for the Functional Requirements Document, System/Subsystems Specification, Test Plan, and Test Analysis Report which are described in FIPS PUB 38 (1). Preceding each proposed security section is a brief narrative explaining the rationale behind its content. Determining the sensitivity of a system, and conducting and documenting a risk analysis (5) of a system are pre-requisites to the preparation of the FIPS PUB 38 (1) documents and are not covered in this paper.

1.4 Limitations

Documentation of system security in most cases cannot be isolated in a single section of a document. It is not intended that all security details be documented under the separate security sections presented in this paper. These security sections serve to remind the authors of the importance of security and provides for the summarizations of all security considerations as applied to a particular document type.

Length limitations for the papers precluded the author from presenting proposed security sections for all ten documents described in FIPS PUB 38 (1). Four documents, which best demonstrate the threading effect of security, were selected for presentation in this paper.

2. PROPOSED SECURITY SECTIONS

2.1 Functional Requirements Document (FRD)

As pointed out in FIPS PUB 73 (6), Section 6.1, security requirements must be adequately defined before software development can proceed. Therefore, heavy emphasis has been placed on the security requirements in the FRD, making that document the cornerstone for building security safeguards into the resulting system. A proposed security section for the FRD is shown in Figure 1. It is intended that the current Section 5 of the FRD, Development Plan, be changed to Section 6.

---

5. SECURITY.

5.1 BACKGROUND INFORMATION. Provide background information which reflects on the sensitivity of the application. Include justification for the degree of sensitivity placed on the system such as, conformance to the Privacy Act of 1974, protection of critical decision making information, preservation of fair competition in the private sector, protection of saleable information, etc.

5.2 CONTROL POINTS, VULNERABILITIES, AND SAFEGUARDS. Provide a description of each control point, the vulnerabilities at the control point, and the safeguard requirements to reduce the risk at the point to an acceptable level.

5.2.1 Control Points. Describe the points in the system where there is a defined vulnerability which requires specific safeguards. A control point can be located at any interface where data move between two administrative, physical, or technical entities. Control points should be considered in three broad categories: input, process, and output.

a. Input Control Points

(1) Source Origin. Identify where input data will be collected, prepared, and entered to the system.

Figure 1. Proposed Security Section for the FRD

Figure 1. Cont'd

(2) Source Backup. Identify where source data will be collected, stored, and/or destroyed after input to the system.

(3) Data Entry. Identify the personnel positions and the terminals that will be permitted to perform data entry, update, and corrective actions.

(4) Error Correction. Identify the points where data input errors will be detected, reported, and corrected.

b. Process Control Points.

(1) Accuracy and Completeness. Identify the points in the processing cycle where the system should notify the user that input data has been accepted and/or that the requested processing has been completed.

(2) System (Programmed) Interfaces. Identify the points in the processing cycle where data are to be internally passed or retrieved to or from other systems.

c. Output Control Points.

(1) Production. Identify personnel positions and terminals that are permitted to receive output.

(2) Distribution. Identify the steps and personnel positions involved in the distribution and disposition of computer output products.

Figure 1. Cont'd

5.2.1.1 Vulnerabilities. Describe the vulnerabilities at each control point identified in 5.2.1. A vulnerability is a design, implementation, or operational condition inherent in the application or system which lends itself to error and/or loss or compromise of information.

5.2.1.2 Safeguards. Describe the safeguard requirements at each control point to reduce the vulnerabilities to acceptable levels. Consider at least the following areas:

a. Administrative Safeguards. An administrative safeguard is defined as any procedure that requires management oversight; i.e., requires supervision to assure compliance.

(1) Personnel. Identify which personnel positions will require security clearances because of their association with the proposed system.

(2) Distribution. Describe any requirement for a variance from standard distribution procedures.

(3) Constrained User Environment. Describe any requirement to limit operation of the proposed system to certain terminals or periods of the day.

(4) Collection and Preparation. Describe requirements for the proper control of collection, preparation, and backup of input data.

(5) Access/Permission. Describe procedural and safeguard of lists of personnel authorized to access and approve change requests for the system.

Figure 1. cont'd

b. Physical Safeguards. Physical safeguards are defined as any physical means that limit access to data; i.e., locked rooms, vaults, card/key access, and locked doors.

(1) Dedicated Equipment. Describe any requirement for dedicated equipment to aid in maintaining system security. This may include storage media as well as processors or terminals.

(2) Storage and Protection. Describe requirements for onsite and offsite storage and protection of materials (programs, data, documentation, etc.).

c. Technical Safeguards. Technical safeguards are defined as any automated process that assures appropriate processing; i.e., passwords, audit trail reports, etc.

(1) User. Describe any requirement for managing user access.

(2) Process Safeguards. Describe the need for any unique data validation procedures which may provide added integrity.

(3) Describe any unique automated output report labeling requirements to be imposed on the system.

5.3 SYSTEM MONITORING AND AUDITING. Describe user requirements for the production of an audit trail including automated reports or journals necessary to monitor the system.

5.3.1 Journalizing (Event Recording). Describe the journalizing requirements for the system. Journalizing is the recording of selected events as they occur within the system and provides the basis for monitoring the processing and use of data and the use of computer resources.

Figure 1. Cont'd

a. Triggering Criteria. Describe the conditions (functions, events, dates, times, unusual circumstances, etc.) which will trigger the creation of an entry on the automated journal.

b. Identification Information. Describe the identification information, external to the application system, such as date, time, system or function ID, user name, terminal ID and location etc., to be recorded in the journal entry.

c. Application data. Identify the application systems data to be recorded for each type of journal entry.

d. Investigation. Describe the procedural and management requirements for review and follow-up of the journal.

5.3.2 Audit Trail. Describe any additional user requirements for an audit trail. These requirements include such values as total transactions processed by location and time; total records added, updated, and deleted by location and time; and various dollar totals. As an example, financial accounting systems must comply with GAO requirements for an audit trail from source documents through the system to outputs.

2.2 System/Subsystem Specification

The threading effect of security in the System/Subsystem Specification is accomplished through backward reference to the FRD. A proposed security section is shown in Figure 2. Note that in paragraph 5.2 of Figure 2, all security requirements defined in the FRD must be addressed. It is intended that Subsection 3.4 be deleted from the current document and be replaced by a separate Section 5 covering security. The current Section 5, Program Specifications, would then become Section 6.

---

5. SECURITY

5.1 SECURITY REQUIREMENTS. Summarize the overall security and privacy requirements imposed on the system/subsystem.

Include:

a. Privacy Act Compliance.
b. Data Security.
c. Data Integrity.
d. Access/Update Authority Controls.
e. Privacy Act Disclosure Accounting.

If no specific security requirements are imposed on the system, state this fact.

5.2 SECURITY SAFEGUARDS. Describe or summarize the security safeguards included in the systems design and how they satisfy the security requirements. If a safeguard has been described as an integral part of the system logic in other sections of this document, reference should be made to the appropriate sections. When appropriate, identify the individual programs in the system which are involved in satisfying a particular safeguard requirement. All safeguard requirements described in Section 5.2.1.2 of the FRD should be addressed in this section as follows:

a. Administrative Safeguards. An administrative safeguard is a procedure that requires management oversight to assure compliance. Describe the administrative safeguards which have been included in the systems design. If an administrative safeguard falls outside the systems design, describe any assumptions made about the safeguard and how the designed system will interface with the safeguard.

Figure 2. Proposed Security Section for the System/Subsystem Specification

Figure 2. Con't.

b. Physical Safeguards. Physical safeguards are defined as any physical means that limit access to the data; i.e., locked rooms, vaults, and card/key access. Describe the physical safeguards which have been designed into or are required by the system. If a required physical safeguard already exists or the responsibility for its implementation falls outside the application system development, describe any assumptions made about the safeguard and how the system will use the safeguard.

c. Technical Safeguards. Technical safeguards are defined as any automated process that assures appropriate processing, controls access to the data, or provides audit trail information. Describe where and how technical safeguards are to be implemented in the system.

2.3 Test Plan

It is important that testing of all security capabilities be included in the test plan. The proposed security section, shown in Figure 3, requires that all security requirements and capabilities described in previous system documents be addressed, even if they are excluded from testing.

---

5. SECURITY

5.1 SECURITY SAFEGUARD TEST DESCRIPTIONS. Summarize the security safeguard requirements imposed on the system and specifically identify the test(s) described under Section 4 of this document which will demonstrate the ability of the system to satisfy each security safeguard requirement. Identify required safeguards which cannot be tested within the scope of this test plan. Provide an explanation concerning the responsibilities for testing these safeguards or why testing is not necessary. All security safeguard requirements described in Section 5.2.1.2of the FRD and/or Section 5.2 of the System/Subsystem Specifications should be addressed in this section. If security requirements, functions, and tests have been listed under Sections 3.1.1, 3.1.2, and 3.1.3 of this document, reference should be made to those sections.

Figure 3. Proposed Security Section for the Test Plan

2.4 Test Analysis Report

To support system certification, a requirement of OMB Circular A-71, Transmittal Memorandum No. 1 (2), it is necessary in the Test Analysis Report to break out security in a separate section. This will allow for easy review of the report to determine if the system meets previously established security requirements. A proposed security section is shown in Figure 4.

---

5. SECURITY

5.1 SECURITY TEST SUMMARY. Summarize the security capabilities which were included in the systems test and itemize the specific security deficiencies detected during the conduct of the test. The results of the individual tests, system findings, and a thorough analysis of deficiencies along with recommendations have been covered in Sections 2, 3, and 4 of this document. The portions of these sections which specifically address systems security should be referenced in this section. If no deficiencies were detected during the systems test, state this fact.

5.2 SECURITY TEST ANALYSIS. Based on the results of the systems test, provide a statement concerning the adequacy of the system to meet overall security requirements as described in Section 5 of the FRD and Section 5 of the System Test Plan.

Figure 4. Proposed Security Section for The Test Analysis Report

REFERENCES

(1) National Bureau of Standards. Guidelines for Documentation of Computer Programs and Automated Data Systems; 1976 February 15; FIPS PUB 38.

(2) Office of Management and Budget. Security of Federal Automated Information Systems; Circular A-71, Transmittal Memorandum No. 1.

(3) General Services Administration. Federal Property Management Regulations; Security of Federal ADP and Telecommunication Systems; August 11, 1980; 41 CFR Ch. 101.

(4) Office of Personnel Management. Federal Personnel Manual System, Personnel Security Program for Positions Associated with Federal Computer Systems; November 14, 1978; FPM Letter 732-7.

(5) National Bureau of Standards. Guideline for Automatic Data Processing Risk Analysis; 1979 August 1; FIPS PUB 65.

(6) National Bureau of Standards. Guidelines for Security of Computer Applications; 1980 June 30; FIPS PUB 73.

USING FIPS PUB 38: A PRACTICAL EXPERIENCE

Patrick O'Conor
Science Management Corp.

Samuel T. Redwine, Jr.
MITRE Corp.

ABSTRACT:

The paper describes the experience of the authors as they established a technical writing organization for the purpose of preparing computer software documentation with the FIPS PUB 38 Guideline. The evolution of the documentation group within the corporate context is presented along with the methods that were developed for the production of documentation. The pervasive way in which the Guideline influenced the operation and conformation of the group's procedures is discussed. Problems the authors encountered with the use of a standard derived directly from FIPS PUB 38 and the solutions evolved to counter them are shown.

Keywords: Documentation; Structured Interview; Technical Writing; Documentation Guidelines; Documentation Procedures; Case Study; Documentation Organizations.

PREFACE:

The authors organized a technical writing group for the purpose of preparing computer systems documentation within a medium sized company whose principle product is computer systems software and services. Much of the software built by the company was subject to requirements to be documented according to the U.S. Dept. of Energy 78-6 Documentation Standard, a derivative (almost verbatim) of the FIPS PUB 38 Guideline. We will describe the development difficulties encountered, and the detailed documentation methodology as it evolved. The paper will show the pervasive way in which the standard influenced the formation, and operating procedures of this technical writing group.

The group was judged to be highly successful as was evidenced by verbal and written expressions of satisfaction from its clients, and by issuance of praise from higher management. Problems of quantity and quality in using FIPS PUB 38 for the production of computer systems documentation had to be solved, and due to techniques described in this paper, this group was able not only to produce technical documentation at substantially reduced costs, but also to free scarce systems staff personnel for roles in which they were better suited and utilized.

1. THE STARTING POINT:

In the late summer of 1979 one of the authors (S.T.R.) became the new manager of a forty person systems staff providing contract systems and programming support to the Energy Information Administration (EIA) for a variety of users in the Dept. of Energy (DOE).

He found the following situation: the systems staff had grown from less than ten to forty persons performing a dozen concurrent, independent projects under four project managers. The contract, a task order type, called for separate budgeting and scheduling for each task order. Documentation for the systems produced under each task was required to conform to EIA documentation standards. The EIA Standard (referred to by its issuance number 78-6) is a typographically poor copy of the FIPS PUB 38 Guideline.

Much work had been done on a code first, document later basis. This had frequently resulted in documentation being omitted altogether. When the author first began to manage the group, no examples of good documentation prepared by the systems staff and using the guideline could be found despite a sincere search effort. This search had included the client organization that had mandated the use of the standard. Despite this, documentation was found to be of great importance to DOE client opinion since it was the most visible evidence of the existence of their systems. Lack of documentation, and bad documentation, had been a frequent cause of complaint against DOE technical managers by the DOE clients for whom the applications were developed.

It therefore formed a major (perhaps the single largest) determinant of client perception of corporate and systems building competance.

Identification of the need for the technical writing group arose from the inability of our systems staff personnel to prepare documentation according to the DOE Standard. This inability to cope with the Standard came from two sources; the lack of writing ability among the systems people, and the lack of understanding they had in trying to interpret the Standard. The typical response of the systems people when confronted by the need to prepare manuals using the Standard was to complain of its paradoxical imposition of requiring specific rubrics to be used which were themselves subject to interpretation. Parkinson's Law was much in evidence in this, and systems staff were loath to expend creative energy in the direction of documentation, indicating most often that theirs was a higher mission in life, i.e. programming and systems analysis.

Using the 78-6 Standard often required determining what the Standard meant. Even knowledgeable systems people who were well acquainted with the vocabulary used in 78-6 were at a loss to understand its intentions. Most of the systems staff could not write readable English in any event, and most project managers spent extraordinary amounts of overtime editing and rewriting. But, the project managers themselves were not professional writers or editors. The documentation efforts thus produced were not only very poor, but were being written with the company's most expensive resources. Their documentation efforts impacted their systems development activities and were contributing to major delays.

2. EARLY HISTORY:

The beginning of the technical writing group occured when it was recognized that the efforts of the systems people were fruitless in this activity. A technical writer was recruited largely to off-load the burdens of our project managers. The person hired, while familiar with data processing and a sound writer, had no experience writing D.P. documentation or with the DOE 78-6 Standard. This single technical writer could not begin to meet the large demand for documentation manuals. Training, role guidance, and an organized relationship with the systems personnel were not provided. This, and the unrealistic demands of our project managers created difficulties for the writer. All of the project managers demanded technical writing time on arbitrary schedules which were typically an order of magnitude too short a time frame to allow production. They dismissed in an offhand manner any suggestion that this caused difficulties or that documentation was a rigorous aspect of the computerized environment demanding respect in its own right.

A few weeks after the hiring of the technical writer, a Quality Assurance (QA) Manager (author P.O'C.) was brought on staff to address software quality issues, establish QA in the systems staff environment, and to bring better order to the process of systems construction. Initial QA efforts were aimed at the creation and installation of programming standards, and the procedures involved in processing systems task orders from DOE clients. This led to QA review of systems staff products including documentation. These reviews pointed to the need for reform of the management approach to documentation production and also to reform of the quality of the documentation which was still largely being produced by the systems staff.

A second technical writer was recruited. It was clear however, that more sweeping changes were required in the production of documentation. This was punctuated by a particular task: the "Lloyds Shipping Index System" (LSIS) which entered the local corporate mythology as the "Lloyds Crunch". (The Lloyds of London Insurance organization supplied certain information for DOE data bases on oil tanker traffic.) The project had slipped several due dates, generated copious amounts of overtime, and had created gross image problems for us with respect to our client.

The LSIS system had been finalized and the documentation for it, which had been produced by the systems staff, was presented for quality assurance review less than three days before the scheduled delivery date. The documentation proved to be disastrously poor; and was made all the more pitiable given the agonized overtime labor of the systems personnel that had been used to produce it. Most rubrics were marked "not applicable", while others had incoherant single sentence explanations that seemed to bear little reference to the definition for the rubric in the DOE Standard.

An embarrassing renegotiation with the client on the due date and a massive effort on the part of the QA manager and technical writer were required to prepare a completely new set of manuals. The result was an acceptable deliverable that solved the critical situation created by the Lloyds task, and provided the first fledgling example of what our 78-6 Standard documentation should look like.

During the strenuous effort involved in producing the Lloyds documentation on an extremely tight, last chance schedule, resolve developed among the participants and management not to repeat the mistakes of the past. The QA manager prepared a long memo proposing a new approach to the preparation of documentation employing a separate technical writing group organized on a par with the systems staff. The memorandum presented cost, organizational, and quality arguments. The response of our higher management was skeptical approval, and we began to recruit writing staff and move toward a better documentation production approach.

Documentation improvement efforts required not only internal changes, but also client education. The client personnel involved in the different development tasks had varied backgrounds and many had never seen good Standard documentation in their federal environments. Most wanted less documentation than was required by the contractual necessity to conform to the DOE Standard. They did not want to pay for it and had a philosophy that every dollar not spent on programming was wasted. Substantial pursuasion efforts were required.

3. DOCUMENTATION PRODUCTION:

There are several key concepts in our production methodology. We do not employ technically trained writers (i.e. programmers) to prepare technical documentation. We have developed and tested in practice a set of simplistic, almost mechanical, techniques to aid in the production of standardized manuals; and have provided these "tricks" to people with demonstrated writing ability. The "tricks" and the people have in turn been embedded in a distinct, controllable, documentation production procedure. Overall the results were striking. We achieved an order of magnitude increase in the productivity factor for documentation. We also freed many systems staff labor hours, and this directly resulted in a marked decrease in systems due date slippages, and other problems.

The fundamental premise underlying our approach is specialization of labor. It proposed,

> "let systems staff build systems,
> and let writers write documentation."

The common sense of this proposition given our experience with having programmers write documentation was inescapable: it was a gross mistake to force the systems staff to write manuals since they disliked writing, did not comprehend (or wished to comprehend) the 78-6 Standard, and could not write English in any event.

3.1 The Documentation Organization:

The organization was composed of five technical writers reporting to a senior technical writer. The group was placed under the QA Manager who also functioned as the Documentation Manager. This organization had two noteworthy features. One is the position of the group under the quality assurance function. The second is the existence of a senior technical writer in addition to the group manager. The group's organization under quality assurance imposed a great concern for the quality of the documentation products. This was true not only of the mechanical aspects of documentation quality, but was extended with special emphasis to the writing: the English itself. Finding itself with the necessity of producing written products, the QA function was determined that no one would ever have a legitimate opportunity to complain of poor quality documentation. Rigorous requirements for writing, review and rewriting were imposed. The group's de facto motto became, "rewrite is a way of life."

The role of the senior technical writer emerged as an administrative necessity. The demands of scheduling, review and rewrite, record keeping, planning and estimating required that one of the writing staff be given an administrative role in addition to writing duties in order to shift time demands from the QA/Documentation Manager. This was also part of a goal in training to ensure that an additional person knew how to perform the production scheduling and other adminstrative functions for the group in the absence of the principal manager. The staff of the technical writing group never exceeded seven persons. There was little need for further suborganization. However, we recognize that separating docu-

mentation management from QA would provide desirable organizational independence during reviews.

The people who comprised the technical writing group had diverse backgrounds, but several obvious common characteristics. Their professional diversity spanned from public administration to english literature, their personal diversity from genealogical research to macrame', but their commonalities were few and easily identified. They were all;

- o intelligent and well educated,

all had graduate degrees, or had attended graduate school. One had a Phd. They could all;

- o write well at the outset,

and their writing improved from an initially high standard with limited training time, on the job practice and writing review. With a single happenstance exception;

- o none had any training or experience in computer systems, all expressed skepticism about their ability to succeed in the role of technical writer.
- o All were aggressive, and were eager to try, and to learn.

In the setting that developed for documentation there were particular requirements that arose for the role of the documentation manager. The role of this manager demanded that he possess writing skills, be able to recognize good writing from bad, and be able to identify writing talent in others. The manager's duties included;

- o Evolution of the procedures (performing systems analysis on the methods of producing documentation),
- o Writing staff training,
- o Review, and rewrite,
- o Writing staff recruiting,
- o Documentation cost estimating,
- o Production scheduling, and
- o Maintenance of intracompany relationships related to the production of documents by interfacing with systems staff.

The utilization of non-technical personnel to write computer systems materials requires that they have easy and immediate access to someone who can explain concepts and provide daily guidance on technical matters. It is a key point of productivity to recognize that only one such person is required: not all members of the writing staff must be, or should be, systems experts. Simple dollar/word cost efficiency precludes such an approach. It is not an absolute requirement that the systems expert in the writing group also be the group's manager, although it was the case in this instance, and worked well.

3.2 The Documentation Process:

A step by step procedure was thought out for the process of preparing documentation according to the 78-6 Standard. The utilization of non-technical personnel for writing of this type required that a standard in outline form be employed. Had it not been that a standard was contractually required, or if none were available, it would have been necessary to create standard outlines due to the nature of the documentation process we developed. It would not have been possible to routinize a production procedure, perform meaningful cost estimates, or gather proper statistics if the product had not been standardized.

The standard to be used may be arbitrarily chosen, but meaningful comparisons and planning are only possible among products of similar conformation. The methods that were evolved for the gathering of technical information for the preparation of text were only usable if guided by a detailed outline of the required information. The 78-6 Standard provided such an outline. We found, in our prior adverse experience, that writers cannot gather material or correctly organize it, without the guidance of adequate documentation standards.

Our principal method of gathering technical material for preparation of

manuals is the "Structured Interview". The structured interview consists of tape recording verbal discussions of the systems to be documented during interview sessions guided by the manual outlines provided in the Standard. The technical writer as interviewer quizzes a systems staff interviewee by reading and discussing the rubric definition from a copy of the standard outline in a conversational manner. The interviewee is asked to discuss in his own words the information called for in each rubric. The interview proceeds, rubric by rubric until the basis for a complete manual is recorded.

Even the most complicated systems rarely took more than a single, forty to sixty minute interview per manual. This was found to be adequate for the production of most manuals. The verbal communications skills of the technical writers and systems staff varied, and this impacted the time required. Interviewing was not a mechanical process, but was subject to the nuance of meaning of each rubric in the Standard. Moreover, the interpersonal skill of the interviewer influenced the quality and quantity of the information obtained. The involvement of systems staff consisted of; participating in these interviews, providing miscellaneous supplementary materials such as flow chart sketches, and reviewing draft manuals for the accuracy of technical content. The net result was that systems staff spent nearly all of their time on systems work.

The invention of the structured interview took advantage of several human traits. It was found that even the most introverted staff member loved to talk about the cleverness he had built into his COBOL, or TSO CLISTs. In the process of verbalizing, the natural order of ideas concerning the system would flow out: each idea queues up the next in its natural sequence. The tape recorder faithfully captures this "stream of consciousness" and the outline from the Standard gives it directly the rough form of a verbal FIPS PUB 38 manual.

The device of employing a tape recorder also frees the mind of the interviewer from the tedious business of note taking, ensures that nothing is missed, and allows the writer/interviewer to listen in detail to what is being said. The technical writer announces each rubric from the Standard outline by name and number prior to the discussion of the topic so that the tapes are automatically organized into the format of the final written product, and a rough form of sequential search capability by physical sequence of taped rubric numbers aides the writer in finding and reviewing interview material. The tape cassettes form a record that is reviewed as many times as needed by the technical writer. The technical writers would listen to these structured tapes, absorb and digest the verbal material, and then prepare a first draft.

Another key concept in the process of preparing technical manuals was the realization that there are different types of English. In this context there are three types; spoken English, ordinary prose English, and technical prose English. The main function of the technical writer is to be a converter of verbal English, as found on the stream of consciousness tape recordings, into ordinary prose, and then via rewrite, into technical prose.

The process of writing a standard FIPS PUB 38 manual was broken down into a sequence of thirteen discrete steps. Each step consists of an activity having duration in time which is marked by its completion date. Each step is performed by a single individual who is held responsible for its on-time completion. The steps are;

	Activity	Responsible Person
1.	Structured Interview	Tech Writer
2.	First Draft Writing	Tech Writer
3.	First Draft Typing	Secretary
4.	Content Review	Systems Staffer
5.	1st QA Review	QA/Documentation Manager
6.	Draft Revision	Tech Writer
7.	Revision Typing	Secretary
8.	Proof Revisions	Tech Writer
9.	1st Delivery to Client	Tech Writer
10.	Revisions Per Client	Tech Writer
11.	2nd QA Review	QA/Documentation Manager

12. Revision Typing/proofing — Secretary

13. Final Delivery — Tech Writer.

This sequential, breakdown of the process formed the basis for the system of management controls devised to guide the production of documents. Only two forms are needed for management control. The first is the "DOCS PLAN", (see the figure). This form is used for; estimating, production scheduling, production tracking, and cost statistics recording. Space is provided at the top of the form for recording qualitative parameters of the system being documented. The second form needed is a simple calendar resource matrix arraying each technical writer and other production resources versus the business date. The calendar matrix is used to allocate technical writer staff time to projects in process.

A DOCS PLAN form is kept for each project and has space for the ten FIPS 38 manuals as well as a Task Management Plan, and non-standard documents, for example proposals. One calendar resource matrix is kept for the technical writing group as a whole for all work in process. The calendar resource matrix is the integrating control mechanism by which the writing activities for many separate projects are coordinated. Maximizing the use of resources requires that project scheduling be coordinated across all projects. Production schedules are prepared as many months in advance as is necessary for demand to allow for proper workload planning, and staff recruiting.

Use of the DOCS PLAN form allows the compilation of reliable statistics on the resources required for each set of manuals. These statistics have been very useful in the preparation of documentation cost estimates, and in production scheduling. No project that has ever been processed using these methods has ever exceeded its budget or missed even a single, intermediate step production date, much less a client deliverable date. This can be attributed to copies of the production schedule being provided to the persons who were to be held responsible for each step. The due dates established are for small steps that are individually coordinated with the person's other activities, and the staff hours allowed for each activity are based on well accepted empirical data. Further, each individual in the production process is asked to review the production schedule for reasonableness, and is given the opportunity to request revisions in the schedule prior to being held responsible for it.

Project production scheduling has been found to be facilitated by early participation of the technical writing staff and management in each systems project. Where resources for technical writing are constrained by staff limitations, and there are many projects competing for those resources then "1st day" participation by the documentation manager is indispensable to smooth operations.

3.4 Quality Review and Rewrite:

Reference to the production steps shown above reveals that six of the thirteen steps are quality assurance in nature. Reviews are performed by the project systems staff, the QA/Documentation manager, the technical writers and the client. During each review rewrite occurs. This rewrite is performed only by the writing staff based on marginal comments by reviewers. This is in recognition of our premise to specialize staff roles, and on negative experiences with having the systems staffers attempting to perform rewrite. During its lifetime the technical writing group has never had a manual rejected by a client, nor even had any serious negative comments.

During the training of our technical writers we communicate that rewrite will always occur and that it should not be considered as being critical or threatening in nature. Rewrite is viewed instead as being a part of the necessary process of corporate writing.

3.5 Transition Activities:

The documentation group did not come into existence immediately after the writing of the memorandum that suggested the efficacy of creating such a group. Nor did the operational procedures, the concepts underlying the procedures, or an understanding of how the Standard would shape the evolution of these things. In this section we will describe how the documentation group grew up and transitioned from pre-group approaches into something capable of producing good quality, FIPS 38 manuals, in a manner akin to an automobile assembly line.

Higher management made no objection to the foundational proposition, but expressed skepticism about the ability of third party technical writers to document systems that they had not programmed themselves. It was

DOCS PLAN

Page 1 of 2

Task Name: _____ Status As of: _____

_____ Date Planned: _____

SMC No: _____ ETA No: _____ Report/Plan By: _____

SMC Proj. Mgr: _____ Task Start Date: _____

SMC Tech. Dir: _____ Est. Compl. Date: _____

Govt. Mgr: _____ Act. Compl. Date: _____

System Statistics Related to These Docs:

No. of COBOL Programs: _____ No. of Utilities: _____

No. of COBOL Source Lines: _____ No. of JCL Lines: _____

No. of SUPERMYLEUR Macros: _____ New System?____,or Remodeling?____

No. of SUPERMYLEUR Lines: _____ Is Formal DBMS Used? _____ (circle which)

No. of TSO CLISTs: _____ S2K ADABAS TOTAL MARK4 IMAGE Other

No. of TSO CLIST Lines: _____ How Many Programmers Assigned? _____

No. of Data Sets in Sys: _____ Programmer Hours Est: _____ Actual: _____

(EST. DATE / ACT. DATE / EST. HOURS / ACT. HOURS)

DOCUMENT	Est.Pages Act.Pages Writer New / Rev	TW Struct. Interww.	TW 1st Draft	C Type 1st Draft	PM Content Review	QAM 1st QA Review	TW Revise Draft	C Type Revisions	TW Proof Revisions	TW 1st Delivery	TW Revisions Per Client	QAM 2nd QA Review	C Type Revisions	TW Final Delivery	QAM Docs. Mgt.	Total Hours by Type: QAM	TW	PM	C
Task Mgt. Plan																			
Func. Req. Doc.																			
Data Req. Doc.																			
Sys. Specs.																			
Pgm. Specs.																			
DB Specs.																			

DOCS PLAN

Page 2 of 2

Document	Est.Page Act.Page Writer New /Rev	TW Struct. Interww.	TW 1st Draft	C Type 1st Draft	PM Content Review	QAM 1st QA Review	QAM Revise Draft	C Type Revisions	TW Proof Revisions	TW 1st Delivery	TW Revisions Per Client	QAM 2nd QA Review	C Type Revisions	TW Final Delivery	QAM Docs. Mgt.	Total Hours by Type: QAM	TW	PM	C
User's Manual																			
Ops. Manual																			
Pgm. Maint.Man.																			
Test Plan																			
Test Anal. Rep.																			
Total Est. Hours																			
Total Act. Hours																			

Two Page Documentation Planning Form

a subtle but important point that higher management allowed the technical writing group to form, but did not make a clear, affirmative decision to do so, nor did they provide aggressive support in the intra-company growing pains that arose as the writing group developed. This caused unnecessary and exaggerated difficulties with other managers. It was only much later (6 to 8 months) that higher management gave recognition to the formation of the writing group as a key factor in achieving a turn-around in issues of systems quality and client satisfaction; as well as an order of magnitude decrease in systems development problems.

Substantial care was taken in the process of recruiting the technical writing staff. Many more applicants were interviewed than the number of available positions. We recruited over a period of several months in response to our mid term scheduling needs as indicated by our resource matrix. All applicants were required to submit writing samples as a demonstration of their ability. The writing sample was the most important aspect of the screening process. We found that any form of writing; school papers, letters to the editor, etc. were adequate to guage the applicant's writing prowess. Misrepresentation of credentials was discovered to be common. Once an applicant had passed initial screenings all references were carefully verified.

The group was small enough to allow for individual daily counselling on writing style, systems concepts, rubric definitions, and production procedures. Formal classroom sessions were held infrequently due to production needs, and were confined to fundamental computer science concepts. The learning curve was rapid for the people we selected. All were productive within a two week time frame. Three months of experience was enough for each writer to perform the role with full independence and productivity.

4. FIPS PUB RECOMMENDATIONS:

Daily experience with FIPS PUB 38 and its derivative, DOE 78-6, leads us to make several recommendations concerning the pragmatic aspects of using standards. Standards, if they are to be living things, used broadly and thus serving their purpose, must be proletarian in approach, not lofty and understandable only by those with ten year DP backgrounds and advanced degrees. They must also be flexible enough to serve a broad range of applications. And, they must be explicit, or perhaps it would be better to say that they should have a minimal amount of vagueness that gets in the way of ordinary folk trying to use them.

4.1 Proletarian Useability: Provide a Tutorial; Improve Rubric Definitions:

We found that having been mandated with the use of the 78-6 Standard we only had available a copy of FIPS PUB 38, and a copy of the 78-6 Standard (which had been retyped by DOE EIA from FIPS PUB 38) upon which to base our actions. We had to establish for ourselves how the Standard was to be related to the production of documentation. Our early experience showed that it was insufficient to merely hand a potential author a copy of the Standard and command the creation of a set of manuals. We painfully learned that even very capable writers were incapable of employing the Standard without the benefit of daily training and guidance in its interpretation, and in its implied process.

The most frequent question asked by our writers during the initial months was, "What does this rubric mean?" Sometimes even the long term DP veterans who were managing the technical writing group found this very hard to answer. In the process of providing the answers to our staff's questions it is clear that we were redefining the Standard for the exigency of the moment. I.e., we were confronted with the situation of not what does the Standard mean, but rather what did we say that it meant. Substantial amounts of personnel dollars were involved, and firm decisions had to be made concerning the Standard's requirements, or else we would have suffered the costs that vagueness directly created in additional rewrite and reduced quality of written communication.

We see this as a flaw in the Guideline, not as an aspect related to its attempted flexibility. We found the Standard to be very hard to use, and it is clear that this causes it not to be used as widely as it might. We would not have used it had we not been contractually required to do so. We would have devised our own outlines.

This leads us to make two recommendations on the point of usability; (1) provide a tutorial as a supplement to FIPS PUB 38 containing recommended guidelines on the use of the Guideline in the production process of preparing manuals; and (2) provide improvement in the definitions of the

rubrics in the Guideline itself (better developed, more elaborate, and more clearly worded). It is clear that the writers of FIPS PUB 38 went too far in the direction of flexibility and generality at the expense of useable clarity.

4.2 Redundancy:

FIPS PUB 38 describes a ten manual set. Much of the material in this set is redundant; e.g. the general information sections, much of the data base description material, and other information is common in all manuals of the set, or is found under many rubrics. This is inappropriate for most application systems of small and moderate size (i.e., the vast lion's share of the project count). This tends to encourage "xerox machine" authorship which contributes nothing to the usefulness of documentation.

The Guideline should be revised to permit, as an option for more modest systems, that the ten manuals be considered as chapters in a single manual wherein the material which is now redundant would appear only once. An alternative outline should be given as a second part to the Guideline which shows the rubrics for a single manual documentation approach with chapters that correspond to the concepts for each of the current manuals in the Guideline. This would have the effect of encouraging the application of the Guideline to smaller systems with a concomitant reduction in the cost of documentation.

4.3 Documenting Interactive Systems:

We have found that the current version of the FIPS PUB 38 Guideline is very awkward for use in documenting systems which are partially or wholly interactive in nature. FIPS PUB 38 has a consistent batch flavor throughout notwithstanding that specific references to batch processing are not found. We recommend that the Guideline be revised with a view towards the interactive systems case.

4.4 Propagation of the Guideline

In addition to the comments on the content of the Guidelines our experience leads us to make recommendations concerning the establishment, propagation, and use of the next generation of the Guidelines. The new guidelines will be at least as much an act of innovation as it will be one of codifying existing practices; as much a problem of technology transfer as one of regulation. The use of the new guideline will become a contractual requirement for a very large segment of the systems industry and therefore carries substantial cost implications for the government if these aspects are not addressed in the guideline itself.

Accordingly, the establishment of the next generation of FIPS documentation guidelines will not (we think), consist of adopting an existing standard as was FIPS PUB 38. Since significant innovation must necessarily be involved, consideration should be given to prototyping, trial-use periods, major experimental application work, beta testing, or other such methods in addition to the normal draft, review and ballot process. We in the computing profession should be aware that it is very difficult to get a major product correct the first time it is released. Actual performance may frequently vary from our expectations in surprising ways. It is therefore appropriate to anticipate a review and rewrite cycle for the new guideline.

Propagation of the new guideline should not consist simply of publication, and an introductory workshop, but should include a broader range of technology transfer aids over the lifetime of the guideline. Our experience does not point to all the things that should be done, but it does point to a number of needs and suggests a partial set of answers.

Among the types of introductory assistance that a guideline needs are; initial demonstrations of feasibility, detailed explanations of the guideline, and particularly examples. One such approach is found in the IEEE Atlas Language Standard and its accompanying book of good practices. Another example may be seen in the "systems development methodologies" offered by some commercial documentation products.

A key point of our experience is that a set of rubrics with content definitions is not enough no matter how clearly written they may be. Guidance is needed on all the essential components of documentation production including; process, planning and control, estimation, and quality assurance. All of our recommendations point to a need for an expansion of the products and services provided by NBS if good documentation is to be a normal occurrence rather than an exception.

An Overview of the Department of Defense Automated Data Systems Documentation Standards- an Adaptable Standard

Robert R. Hegland

Department of the Navy
NARDAC WASHINGTON DC[1]

This paper describes the contents of the Department of Defense Automated Data Systems Documentation Standards(DoD Standard 7935.1-S). This standard is currently being used by the Army, Navy, Air Force, several defense agencies, and by the Organization of the Joint Chiefs of Staff. A slightly different earlier version served as the point of departure for a federal documentation guideline which was published in 1976. In addition to describing the standard, this paper will discuss some of the management and technical options that may be used while still conforming with the standard.

Keywords: DoD standard, Management options, Document types

1. INTRODUCTION

The Department of Defense(DoD) documentation standard [2] is widely used, has a proven successful record of use, and is flexible enough to be used across the wide range of hardware and software types used in DoD. The last printing, to satisfy specific user requests, was for over 35,000 copies. It is used by Army, Navy, Air Force, Marine Corps, the Office of the Joint Chiefs of Staff, the DoD agencies and also by the contractors that support them in various development efforts. It was intended for the documentation of management information systems but has been successfully applied to other types of systems. Many federal and commercial organizations have also obtained copies. It is used for documenting large and small systems. It is used for documenting systems installed at only one site and those installed at many sites.

2. BACKGROUND

2.1 Need for a Standard

When the predecessor of this standard was developed in the mid 1960's, it was to satisfy the requirements of an environment where several different computer sites were to receive the same software but each site wanted different documentation delivered with the software. Since this documentation would have been very expensive to develop and to maintain, a decision was made to develop a standard that would satisfy the information requirements of the different sites but would also allow the users sufficient flexibility to satisfy their actual unique needs that had originally caused them to request different documentation. During the development of that standard and throughout the several revisions to it, there have been several principles that have been followed by the people involved in maintaining it. Some of these are listed below.

a. Necessary user extensions of the information to be documented must be allowed.

b. The management options and types of flexibility that a user of the standard can exercise in adapting the standard to a particular environment while still meeting the intent of the standard should be discussed in the standard.

c. There should be enough information detailed in the standard to give the person who is trying to follow the standard a good understanding of what the document should contain.

d. The standard should not use terms or phrases that are unique to current philosophy or techniques, such as structured programming, top-down design.

e. The standard must not be limited to specific hardware or software.

f. The use of computers or word-processing equipment to produce the documentation should neither be required nor prohibited.

g. Locally-developed forms should be allowed.

2.2 Terminology

There are three terms that are used herein that need a brief explanation. A "project" is considered to be a development effort that begins with the identification of a need to write a computer program, such as a payroll system, and extends through the necessary analysis and design to programming, testing and placing the program into an operational environment. The size of a "project" varies widely but the standard discusses a way to determine how much documentation is needed for projects of different sizes. The standard outlines ten "document types" that specify what "object documents" should contain. The "document type" is one of the ten documents outlined in the standard, such as a Users Manual or Program Specification. An "object document" is a document that describes a particular project and discusses how, for example, the payroll system is to be designed(Program Specification) or how the user is to use the payroll system(Users Manual).

2.3 Development of the Standard

This standard was developed initially to serve a limited but diverse audience. Subsequently, others beyond that audience found that they could also use it. Copies were made available to many organizations and groups and eventually a modified version became a DoD standard. At each step during the development process different committees reviewed the standard and made changes to ensure that it would be useful to the groups that they represented. While it has not lost its original integrity, it has been improved by the contributions of each of the people who have been involved in its development. It also has provided the basis for several other uses. One individual copied large parts of it, added some other information, and published it as a paperback documentation standard. It was also the primary point of departure for the development of the NBS FIPS Pub 38 [3] that was adopted as a federal guideline in 1976. FIPS Pub 38 has the same document types but their description has been condensed. Essentially the same life-cycle example is also used. In addition to the evolutionary modifications that occurred as the standard was adopted by different groups, there is a DoD task group charged with responsibility for reviewing and improving the standard as well as with responding to any reported problems in its use. In other words, this is a dynamic standard that is maintained to keep up with the state-of-the-art while not limiting it to any particular group of users.

3. STRUCTURE OF THE STANDARD

The standard is a relatively small document of about 150 pages and is prepared using a standard typing format rather than by typesetting so that it looks like the object documents that will be produced from using the standard. This makes the writing process somewhat easier for the authors of the object documents. The standard is made up of three parts which include the information described in the following paragraphs.

3.1 Part 1-General

This part includes short sections on the purpose, scope and objectives of the standard as well as a description of its organization.

3.2 Part 2-Document Development Guidelines

This part includes a discussion of how the standard is intended to be used, how to plan the preparation of documentation, and other items that the manager of a development effort should know about the standard and about documentation in general before writing the first

word of an object document. This part also includes a discussion of how the various standard document types can relate to the development life cycle of an application development effort. The example of a life cycle that is included is generalized, is not a standard itself, and is shown for illustrative purposes only. A discussion of the purpose and general contents of each of the different document types is then presented. There is also a discussion of the options that need to be considered in using the standard. These include the relationship of project complexity to the document types needed to support a particular development effort, how to measure project complexity, and how to decide which of the ten document types are needed for that project; the sizing of documents and the sections within each document type; how to tailor the document types for specific implementations by the addition or deletion of paragraphs, sections, and other information that is called for in the standard; and the use of graphic charts and forms.

3.3 Part 3-Document Standards

This part of the standard includes the ten different document types. Each document type is presented in a way that allows it to be removed from the overall standard for use by the person or group preparing the object document. The paragraph numbers and titles in each document type are exactly as they should be in the object document except as may be modified by the application of an allowable management option. Within each of the paragraphs is a detailed narrative description of the type of information that should be included in the object document. Many of the paragraphs in the document types also have "shopping lists" of items that may be appropriate to discuss in the related paragraph of the object document. That document may present the required information in a narrative, in a figure, or on a form. If a figure or form is used, the standard paragraph number and title must be used but the short narrative text can simply refer to the appropriate figure or form. In a few cases there are examples of figures that can be used to support the text but these usually are avoided in the standard since most sites have their own forms or their own techniques for presenting figures.

4. USING THE STANDARD

4.1 The Document Types and Life Cycle

Figure 1 shows a generalized life cycle and how the ten document types in the standard relate to the phases and stages shown. The transition from one phase to another is usually not as sharply defined as might be interpreted from the figure. The purpose of the document types should be apparent from their placement in the life cycle and from their titles. Even though the standard specifies the contents of ten document types, all ten are seldom produced on a project. The number of document types needed to support the development effort depends on the complexity of the project, on the experience level of the developers and users, on whether or not the developers have worked in the application area where the program is to be developed, and on other managerial and technical considerations.

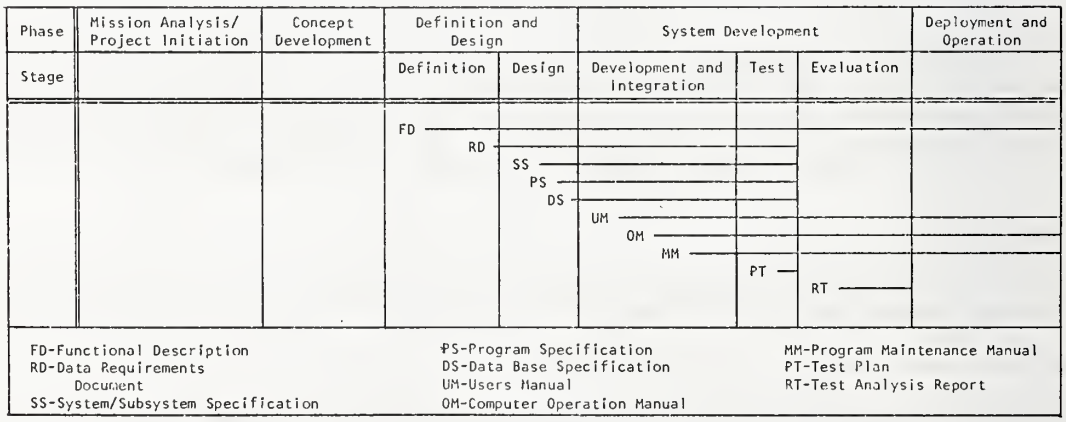


Figure 1. The ADS Development Life Cycle Related to Document Preparation and Use.

4.2 Project Complexity and Document Types

The standard provides a guideline for determining what documents may be needed for a particular project. There are 12 factors listed including span of operation, criticality, and programming languages used. Each has a possible weight of from 1(for a weight indicating limited or easy) to 5(for extensive or hard). When the assigned weights from 1 to 5 are added up for each of the 12 factors, the total is used to enter a table that shows which document types should be written for that project. For most projects of an "average" size and complexity five object documents are usually prepared. The developmental environment may allow fewer or may require more. This is a very generalized guideline to which good technical and management judgment must be applied before deciding what document types will actually be written.

4.3 Audience

Different sites sometimes feel that their organization is unique and so can't use standard documentation. For example, at some sites data retrievals are performed by a manager from a terminal in the working area; at other sites the retrievals may be made by a maintenance programmer. The basic retrieval set-up for the manager might be in a manager's handbook at one site and in a maintenance manual at the other site. Such arbitrary structures would obviously make a standard impossible. To avoid the problem and encourage standards, the DoD standard is oriented toward satisfying the audience by the function that the audience is performing. Regardless of whether the retrieval is being executed by a manager, a programmer, or by a computer operator, the function being performed is one of a user of the data and should therefore be documented in a Users Manual.

4.4 Redundancy

Redundancy is a much discussed topic in writing documentation standards. DoD users have performed studies on the DoD standard to identify any unnecessary redundancy and have found little. What is necessary redundancy? One type is the introductory material needed to identify the purpose of the system, list related documentation, etc. That information serves as "boilerplate" for each object document. Other types of necessary redundancy are "apparent redundancy" and "evolutionary redundancy" both of which are discussed in the DoD standard and which are not undesirable if handled properly.

Apparent redundancy occurs when two document types discuss what appears to be the same information, such as "input" which is discussed in both the Users Manual and the Computer Operation Manual. The information included in the object document should, however, be different for each audience. The user needs information about the structuring of the data input while the operator needs to know how many card decks and tapes to use.

Since project development can cover a long period ot time, the standard allows for including information that has changed since the last document was published. For example, there may have been new equipment acquired that must be discussed in a paragraph dealing with environment since the original, planned description of the environment in the Functional Description. If there has been no change to something that has already been documented and if that document is available to all recipients of the new, updated document, the paragraph can simply refer back to the older document for the information that is still current.

4.5 Document Sizing

The standard provides some guidance to the authors of object documents in terms of planning the overall size of object documents. The standard can be used to write an object document of any document type that might be 100 pages or 500 pages in length and both might conform to the standard. The manager of the project development effort must make it clear to the development staff whether a large or small document is to be written. The tendency to overdocument can be just as bad and as expensive as the tendency to underdocument. In addition, the standard specifies the approximate percentage relationship of each of the sections in the document to the overall page total of that document. This is only a general guide but does help to establish what sections of each document type need to have the primary emphasis.

5. CONCLUSION

Much work has been done in recent years on documentation standards. The DoD standard, from its high degree of acceptance, seems to have bridged the gap between undue rigidity and a totally flexible standard. It has proven itself to be an adaptable standard that can be modified by local managers to meet their needs, is maintained by a committee responsive to general user needs, but is still a standard that provides a useful structure to authors of technical documents. As more tutorial and menu selection systems are written, the standards will need to be reviewed to ensure that these design philosophies can be accommodated in the existing structure or the structure will have to be modified to allow for them. There is still a need to provide the many audiences of the documentation of a project with object documentation that is concise and useful in helping them perform their functions and with documentation that will support the sharing of software.

6. REFERENCES

[1] This paper represents the views of the author and not necessarily the policy of NARDAC WASHINGTON DC or any other Navy command. An earlier version of this paper was reviewed and endorsed by the Information Processing Standards for Computers (IPSC) 0061 committee which maintains the standard.

[2] Department of Defense Automated Data Systems Documentation Standards 7935.1-S, Office of the Assistant Secretary of Defense(Comptroller), Directorate for Automation, Policy, Technology, and Standards, 13 Sep 1977, available from NTIS, Springfield, VA 22161 for $7.50(item PB 272-600).

[3] Federal Information Processing Standards Publication #38 of 1976 Feb 15, Guidelines for Documentation of Computer Programs and Automated Data Systems published by the National Bureau of Standards.

NBS FIPS Software Documentation Workshop, March 3, 1982

SESSION D: Do Existing Standards Work?

Elizabeth C. Weinberger

U.S. Department of Health and Human Services

Summary

Although documentation standards exist (i.e., FIPS PUBs 38 and 64), there are inadequacies within the standards, problems with the way the standards are applied, and often a failure to use the standards. In this session, the panelists and audience discussed various problems with documentation and several ways to reduce such problems.

Ronald Thies began with a discussion of one inadequacy of the current standards which do not contain sufficient security documentation. Due to the increased emphasis on security, particilarly with the issuance of OMB Circular A-71, there is a need for specific security documentation to be included throughout the system's life cycle. Mr. Thies' presentation, "Documenting Systems Security," included detailed examples of security provisions to be added at four different stages of documentation: 1) Functional Requirements Document, 2) System/Subsystems Specification, 3) Test Plan, and 4) Test Analysis Report.

The audience raised a significant question about the potential danger of detailing the system's vulnerable points and identifying the safeguards for each of these vulnerable points. This practice could result in system abuse or misuse. The audience suggested that detailed security documentation have a very limited distribution. Mr. Thies said this isn't such a problem, because although the vulnerable points and safeguards are identified in the security documentation, the specifics of each safeguard are not provided. For example, if a certain vulnerable point in the system required a code to be entered as the safeguard, the actual code would not be part of the documentation.

The second presentation, "Using FIPS PUB 38: A Practical Experience," related how a change in the documentation process, while using the existing standards, greatly improved the quality and timeliness of systems documentation. Patrick O' Connor and Samuel Redwine, Jr. were both employed by a particular company which was experiencing poor documentation products, programmer resistance to doing documentation, management concern over using expensive data processing personnel to write documentation rather than do more programming, and problems in meeting schedules and budget restraints.

The solution was relatively simple. After a major crisis which required a complete rewriting of documentation in an extremely short period of time, the company's top management realized the importance of doing good documentation and raised it as a priority. They then decided to relieve the system staff of this burden and created a division of labor. "Let systems staff build systems, and let writers write documentation." The company hired a technical writer and later expanded by establishing a group of technical writers. They established a process for documentation whereby the technical writer interviewed the systems staff, using a tape recorder and a FIPS 38 checklist.

The changes were dramatic. The technical writers produced solid documentation products, and were less expensive than the data processing personnel. The success generated an improved attitude about documentation and reinforced it as a company priority. And, most important, schedules and budgets were routinely achieved.

The audience had many comments and questions about using technical writers to write documentation. Many commentors echoed the success of this practice, at the Department of Energy and the Navy. Some audience members believed the technical writers should receive data processing/systems training, whereas others strongly disagreed, saying that such training was costly, time consuming and resulted in mid-priced technical writers leaving to seek higher paying data processing positions. The panelists were quite insistent that

the only qualifications for a good technical writer were to be smart, write well and be an "eager learner." This was a key point of the presentation. The panelists claimed that contrary to the belief that a non-technical person can not write such a manual, and can write it well!

The final presentation by Robert Hegland compared the provisions of FIPS 38 with the Depart- Department of Defense's Automated Data Systems Documentation Standard. According to Mr. Hegland, DOD's standard provides more descriptive prose and more examples of documentation than what FIPS 38 provides. Although the DOD standard is quite useful as a general purpose standard and is flexible for the user, the weakness of the standard is that it does not address the newest technology, and does not adequately detail documentation of interactive systems.

In closing, the panelists and audience agreed that the National Bureau of Standards needs to revise FIPS 38 to make it more usable, clearer, easier to understand, and more of a tutorial aid. They suggest that NBS remove the redundancy within the standard, provide examples, and detail the entire documentation process rather than providing just an outline of documents. The new standard should be innovative, allow for prototyping and field testing, and address documentation for interactive systems.

Session E: Proposals for Documentation Standards

Moderator: Trudy Grieb, Hadron, Inc.

Discussants: Edward W. Hurley, Hadron, Inc.
Samuel T. Redwine, Jr., The MITRE Corp.

Recorder: Andrea F. Papilion, Hadron, Inc.

Introduction: In the past, documentation standards have been of very little net value to the audience of documentation (management, project leaders and project staff) and to the authors of documentation. What new approach is needed to produce a new, better set of documentation standards?

The papers in this session proposed approaches to documentation standards for the purpose of overcoming problems encountered in using current -- often inconsistent, sometimes conflicting and awkward -- documentation standards.

The first paper, Proposed Approach to Standards for Documentation of Projects and Systems Based on Actual Requirements, Trudy Grieb, Hadron, Inc., provides an overview of the problems that need to be addressed and possible approaches to solutions.

The second paper, Microcomputer Systems Users Need Better Documentation, Richard A. Bassler, The American University, provides a discussion of problems and vital recommendations for producing better documentation for microcomputer users.

The third and last paper, A Proposed Documentation Standard Based on a System Decomposition and Information Base Approach, Saul A. Zaveler, U.S. Air Force, provides an important proposal to treat the subject of documentation with the same respect and the same concepts of decomposition as those used for the software being developed. The paper also states that considerations for content should be separated from considerations for format and organization.

Approach to Standards for Documentation of Projects and Systems Based on Requirements of the Users of Documentation

Trudy Grieb

Director, Information Systems Technology
Hadron Incorporated
Vienna, Va. 22180

This paper addresses questions about and possible solutions for the documentation of information processing systems and the projects which are the vehicles for creating these systems. This paper (1) defines the problems and requirements of the documentation of projects and systems; (2) evaluates existing attempted solutions (i.e., current documentation standards), and (3) proposes a tested and proven approach to new Federal Government documentation standards based on actual requirements.

Keywords: documentation standards, information processing system standards, project management standards.

1. INTRODUCTION

In the past, documentation standards have been useful to only a small degree and have created unnecessary problems. The purpose of this paper is to: (1) point out the major problems with the past approaches to documentation standards, (2) define the general requirements for documentation standards, and (3) propose a new, promising approach to documentation standards that will be useful to the readers/users, to technical personnel, and to managers, without creating unnecessary problems.

The scope of this discussion includes:

- o Projects for information processing systems;
- o Information processing systems;
- o Documentation for users, line management, project management, and technical personnel.

The intended audience of this paper consists of senior level systems analysts, project leaders/managers, standards analysts/developers, and managers, who are responsible for systems analysis, development, and project management personnel.

2. PROBLEMS

Existing documentation standards for information processing systems are inadequate due to a number of major problems.

2.1 Lack of Context

All existing documentation standards are based on an assumed and/or implied set of concepts and System Life Cycle Methodology (SLCM). Therefore, the assumed SLCM is not clear, and cannot be properly used in the development of a project plan which should include a clear plan and definition of all documentation to be produced. All documentation is produced in the context of a project plan, which is usually different from the one implied by the documentation standard. The authors of documentation then are forced to resolve the conflicts.

Further, the methodologies implied by the documentation standards are usually incomplete, unclear, and/or incorrect. Some of the most common and popular concepts on which most methodologies are based are not workable.

As a result, the current documentation standards do not meet the requirements of the readers/users of the final documents.

2.2 Awkward and Arbitrary Standards

Without a clear, complete SLCM, the developers of documentation standards are handicapped. They need the SLCM as a context for the development of documentation standards. Standards developed without a SLCM are based on arbitrary assumptions and are awkward to follow.

2.3 Lack of Guidance and Definitions

Most documentation standards do not provide sufficient explanations of what information is needed. Part of such explanations must be the clear, complete definitions of terms used in the standard. Another part must be descriptions and examples of required information content.

2.4 Excessive Rigidity

Many documentation standards that have been used over the years are very rigid. They require strict outlines, titles, headings, numbering systems and formats. They do not permit adaptation based on the project and system being documented. They also tend to place primary emphasis on format instead of information content.

3. REQUIREMENTS

The primary requirements of the users/readers of project and system documentation are:

3.1 Control by Upper Management

Upper management must be provided with necessary and sufficient information for decision-making and control.

3.2 Project Management

The Project Manager and project team members must have information which will assist them in achieving the purpose and objectives of the project with efficiency, high productivity, and products of high quality.

3.3 Progress Information

Upper management, project management, and the users must have timely progress information about the project and the system.

3.4 Means of Communication

The Project Manager and project team members must have an effective means of communication about the project and all components of the system.

3.5 Quality and Productivity

Upper management, project management, project team members, and the users need high-quality products and high levels of productivity as well as cost-effectiveness.

3.6 Modern Technology

All personnel concerned with the project and system need to have the standards and final system (including documentation) in harmony with and responsive to modern technology, as appropriate. Modern technology impacts all phases of the system life cycle (SLC), including the project management approach, the system, and the production/distribution of documentation.

3.7 The User as the Customer

The end-users of the system need instructions and descriptive information that will allow them to understand their system and to use it as easily and effectively as possible. They need to have their documentation written and presented in a form and language that they can understand without having to learn about computer and communications technology.

4. CHARACTERISTICS AND RESULTS OF GOOD DOCUMENTATION STANDARDS

Useful and helpful documentation standards must have certain essential characteristics to meet the requirements stated in Section 3 and must avoid the problems stated in Section 2.

4.1 Information Appropriate for Management

Documentation standards should provide for non-technical summary documents which crystalize the major points to be considered and which state the decision(s) requested of management. Such documents will allow management to be informed and to make timely decisions.

4.2 A Context for Document Standards

Documentation standards should be based on a generic, functional, standard SLCM which, in turn, was designed to implement general policies and to meet organizational objectives and goals. Such a standard SLCM provides the required context now lacking and therefore, would allow development of meaningful, helpful, and easy-to-use documentation standards.

4.3 Information for Progress Reporting

Documentation standards should encourage a "document-as-you-go" approach, rather than after the fact. This will allow information to be captured as it becomes available and at a time in which it can meet most progress reporting requirements. This approach will also provide for more effective and reliable communication among project team members in the performance of their work. As a result, project team members will be documenting for their own benefit, as well as for the benefit of management and users.

4.4 Room for Technology

Documentation standards should reflect applicable modern technology. Such standards will encourage, rather than restrict, the use of modern technology in the design of systems and in the production/distribution of documentation. Again, this will increase productivity and quality.

4.5 Appropriate Documentation for End-Users

Documentation standards should describe, teach, and demonstrate the non-technical, straight-forward, user-oriented presentation and language of user documentation. Because very few authors have produced practical, easy-to-understand user documentation, as much guidance as is practical should be included. The resulting increase in quality of user documentation will allow users increased effectiveness of their systems with less effort on their part in learning to use them. In turn, this will encourage increased productivity and decreased costs for the user.

4.6 Clear and Thorough Explanations

Documentation standards should provide clear and thorough explanations of the information requested and why it is requested. Examples should be used freely. This will (1) make the standard easier to use, (2) will teach the inexperienced author what is required, and (3) will permit the standards to be applied in a wide variety of situations and projects.

4.7 Flexibility

Documentation standards should be designed to be enforced at a general level but should be flexible at a detailed level. The standards should not impose an awkward, rigid structure on all documentation that is to be produced.

5. CONCLUSION

The realm of project management, SLCM, procedures, and standards can be compared to the body of a dog. When the documentation standards are the only standards/guidelines that are fully developed and implemented, "the tail is trying to wag the dog". Of course, the dog has great difficulty making progress this way. However, when a SLCM (including concepts and terms) is defined and implemented in its own context of project management policies, the dog has a chance. With appropriate procedures implementing both the SLCM and the documentation standards (and other necessary standards), the dog may actually reach a pre-defined goal efficiently and effectively.

6. RECOMMENDATIONS

The following recommendations are made for the development of documentation standards.

6.1 Provide a Context

First, develop and test a total System Life Cycle Methodology (SLCM) (including a Glossary of Terms) to serve as the context for documentation standards. An SLCM defines what processes, inputs, and outputs are required in a relatively complex project. Such a context and definitions of terms will minimize the usual conflict between what people do to perform their assigned tasks and the information they document about the process they use and the output they produce.

6.2 Define Information

Second, develop and test outlines of the kind of information required by the users/readers of the documentation. In the context of an SLCM, it is possible to develop and distribute workable outlines of information required in each written output. Feedback and discussions will allow the authors of these outlines to test and correct their contents and scopes. Do not include formatting standards at this point in time.

6.3 Describe Information

Third, develop, write and test explanations of the information required (including a glossary of terms) in each part of each required document. These descriptions must provide useful guidance to the personnel who use the standard to produce their documents.

6.4 Develop Criteria

Next, develop evaluation criteria to be used in reviewing the final documents to be produced by all personnel who will use the standards. These criteria (1) will inform the authors what is expected, and (2) will permit an objective review of the final documents by third parties of the final documents.

6.5 Develop Format Guidelines

Finally, develop standards for the organization and format that are required and/or desired in the final documents. Again, this provides guidance to the authors of the final documents.

6.6 Justifications

The justifications for these recommendations are that documentation standards developed in this manner:

- Meet the requirements of the users of the standards
- Simplify the process of following the standards
- Decrease misunderstandings and associated morale problems
- Increase productivity
- Decrease costs
- Increase the quality of final documents.

RECORDER'S COMMENTS

1. Approach to Standards for Documentation of Projects and Systems, Based on Requirements of the Users of Documentation, Trudy Grieb

1.1 Summary of Discussions

This paper presented a systems view of documentation, including requirements, goals, context, and overall functions. It is important to look both at what the documentation is doing now and what it will do in the future.

The question was raised by the discussant as to whether FIPS Pub. 38 should be revised incrementally or whether a radical new approach should be taken. The audience voted about evenly, but pointed out that a middle ground might also be effective.

There are three basic ways to develop documentation in the context of a system life cycle: a thorough approach within the life cycle, prototyping, and a do-it-yourself by the user.

The meaning of documentation includes getting to know what the system is or does. The definition of documentation can range from an extensive manual describing or instructing on a system down to the simple prompts of a program or the code itself. We need to think in terms of a large scale view.

The object of producing standards for documentation is to define a content and format that will make the user happier; but we get little feedback from users on what they want. We develop standards for people producing a system; yet in the past we didn't care how we could make their jobs more palatable.

We are looking at both general project standards and documentation standards. Should these kinds of documentation and standards be separated? We may not know how to produce general standards for all kinds of projects. When documenting a project, we want to know about it while we are doing it. When documenting a system, we need a method of keeping the documentation up to date over the life of the system.

1.2 Summary of Questions

- Q: How would the speaker redesign FIPS Pub. 38, and would there be more or fewer reports in the index?

  A: In redesigning FIPS Pub. 38, a context should be provided. The number of reports is not significant, but the number of sections or bodies of information available at the end of a project or system development is significant. The reports that work best for each project, in harmony with the standards and in context of the project, are the answer.

- Q: Are we talking about a definition of a reasonable basis of theory of what documentation should be?

  A: Yes. We also need to define concepts proven to work versus those that are popular but don't work. The new standards should be based on those concepts that do work.

- Q&A: The speaker agreed that the development of standards should be treated as just as much of a project as the development of software.

A Proposed Documentation Standard Based on a System Decomposition and Information Base Approach

Saul A. Zaveler

Air Force Data Services Center [1]

A proposal is made for revision of FIPS 38 with regard to viewpoint and content. The viewpoint suggested is that system decomposition (into components, functions, subsystems, actions, or events) serve as the principal basis of document organization, and that the principal document types all have similar informational requirements but differ in degree of detail. The informational requirements specify a data base from which the documents are derived. The principal content changes are: identical paragraph numbering for similar information in all documents; provision for interactive systems, and provision for documention of project management matters.

KEYWORDS: Documentation, Documentation Standards, Top-Down, System Decomposition, Software Engineering, FADPUG.

1. INTRODUCTION

1.1 This paper proposes a replacement for FIPS 38 [2], "Guidelines for Documentation of Computer Programs and Automated Data Systems." The intent is to revise FIPS 38 so that it would be better suited to modern methods of system development, and to the documentation of interactive and real-time systems.

1.2 The Federal ADP Users Group, Special Interest Group on Standards and Quality Assurance (FADPUG/QA) conducted a study, in the summer of 1980, on the deficiencies of FIPS 38. One result was a skeleton strawman standard which would serve as the basis of a revised FIPS 38. The study was conducted at the invitation of A. J. Neumann of NBS/ICST. The author of this paper participated in the study and had a principal role in the formulation of the strawman. A discussion of the study has been published [3].

1.3 The author has revised and developed the strawman into an actual documentation standard for experimental use in his organization. This paper reports on that standard. However, the opinions expressed in this paper are those of the author, and are not necessarily those of his employer, or of any other component of the U.S. federal government.

1.4 The proposed standard covers the Functional Description (FD), System/Subsystem Specification (SS), Program Specification (PS), Operation Manual (OM), User's Manual (UM) and Maintenance Manual (MM). It also covers or references the requirements of the Test Plan, Test Analysis Report, Data Requirements Document, and the Data Base Specification. (These documents are defined in FIPS 38.) It also provides for project management information and for additional types of documents.

2. SUMMARY OF RECOMMENDATIONS.

Revise FIPS 38 along the lines suggested by this paper and the draft Documentation Standard which it discusses. System decomposition should be the basis of organization of all prescribed documents. Information requirements should be separated from information display requirements. Provision should be made for documentation of project management

requirements.

Basis for recommendations: FIPS 38 is hostile to modern methods of system development and to documentation of interactive systems. This observation is based on personal and organizational experience and on the FADPUG/QA report cited above. The recommendations will obviate these problems in an effective and enduring manner because they are compatible with a wide variety of system development methodologies, yet do not prescribe any. Documents produced from the proposed standard will be more usable than those prescribed by FIPS 38.

3. GENERIC RECOMMENDATIONS

The recommendations discussed below are key features of the proposed standard. However, they are abstracted from the standard and presented generically. (Standard-specifc matters are discussed in Section 4.) They are all based on personal and organizational experience and on the FADPUG/QA report.

3.1 Separate, at least conceptually, the information content requirements from the information display (actual document) requirements. The required information items can be regarded as elements of a database, the "Comprehensive Information Base" (CIB). Information should be created, captured and collected separately from its display. All documents produced according to the standard are printouts of part or all of the CIB.

Reasons:

(a) Emphasizes compatibility with a wide variety of system development methodologies and life-cycle models. The order in which the elements of the CIB are updated is determined by the methodology and life-cycle model used, not by their arrangement in the resulting documents.

(b) Simplifies the documentation effort. The act of updating a database is more palatible to many programmers than is the act of producing an entire document.

(c) Facilitates documentation in step with the development effort, since only a few items need to be entered into the CIB at any one time as development progresses.

(d) Facilitates flexibility. It should be recognized that documentation standards prescribe reference documents. Other types of documents may be needed with arrangements of information different from that prescribed. The concept of the CIB permits easy correlation between non-standard documents and the information requirements prescribed by the standard.

3.2 Recognize that all documents produced according to the standard have similar informational and organizational requirements; they differ primarily in degree of detail.

3.3 Organize information in all documents along a system decomposition (component) orientation. Inputs, outputs, dialogues, staff actions, and other items should be clearly aligned with the components to which they pertain.

Reasons:

(a) System decomposition along component lines such as functions, actions, events, or sub-systems is characteristic of modern methods of system development. This is true whether analysis is done along functional or data-driven lines.

(b) An action or event orientation facilitates the understanding (for design, use, and operation)
of interactive systems.

(c) The segregation of inputs and outputs from components or functions in the FIPS 38 documents make preparation and use of the documents and implementation of systems from them very difficult (for all but the rarely occurring single function systems). The FIPS 38 organization reflects a hardware orientation which is rarely of primary concern to the elucidation of the system being documented and implemented. Hardware should form the basis of document organization only when it forms the basis of the components.

(d) Facilitates systems analysis. If documentation is done in step with the developement effort, the need for further analysis and the types of questions that should be asked are made very clear through this organization.

3.4 Broaden the scope of the input and output requirements of FIPS 38 to include command, edit and other languages to be developed by the implementers; dialogue and transactions; function keys and special hardware; interactive graphics; and text processing.

Input and output are too cut and dried for the interactive and real-time worlds; the other categories are quite necessary to facilitate and simplify the documentation effort, and increase understandability.

3.5 Encourage graphics in documents such as function charts, data flow diagrams, Gantt and pert charts, Petri net diagrams, decision tables, finite state transition diagrams, and system and logic flow charts.

3.6 Provide for validity of the document. Each document should have provision for discussion of its testing against the actual system being developed or in operation.

3.7 Provide for discussion of management considerations, or reference to program development and configuration management plans.

3.8 Provide linkage to Test Plans and Results of Tests. Such references should appear in all functional areas and anywhere else where the validation of the system is critical.

4. SYNOPSIS OF PROPOSED STANDARD

4.1 The individual items of required information (the data elements of the CIB) are called "documentation topics," and each begins with a prescribed paragraph number and title.

4.2 The documentation topics fall into four broad categories called "sections." Section 1 deals with general information, section 2 deals with management matters, and sections 3 and 4 describe the system requirements. It is frequently convienent to regard a system as made up of components (also called functions, subsystems, actions, events, etc.). Section 3 is used to provide an overview of the system and an enumeration of its components. Each component is described in detail in section 4. Section 3 may also be used to provide information common to many components. If there is an active database administration role associated with the system being documented, then it is suggested that the data items be discussed in detail in section 3, and enumerated where appropriate in section 4. The goal of maximum understandability should determine where information is placed, if the standard permits a choice of section or documentation topic. If

the component approach is not used then section 4 may be omitted.

4.3 Paragraph numbers are of the form

<section or sub-section number>:<topic number>

Components, sub-sections, and topics are all numbered in the manner in which paragraphs are numbered in FIPS 38. (E.g. "3.4.5") Top-level components have level-one paragraphs (i.e. just positive integers). Sub-components have level-two paragraphs, etc. For example, if component 3 has 2 subcomponents, they would be numbered 3.1 and 3.2. (Zero is also acceptable as a top-level component number.)

4.4 Sections 1, 2, and 3 have no subsections. Section 4 has a subsection for each component enumerated in section 3. Section 4 subsections are numbered thus:

4.<component number>

4.5 The types of information to be provided for the system as a whole and for each of its components are the same. Hence the topic numbers for section 3 and each subcomponent of section 4 are identical.

4.6 The table below enumerates the documentation topics in the order in which they are to appear in all documents prescribed by the standard. (They appear in order of increasing paragraph number.) Shown are the paragraph numbers and titles and recommended applicability. As a minimum it is recommended that topics denoted by an "m" be applicable; however, it is highly recommended that topics denoted by an "x" also be applicable. For each documentation topic, the standard provides guidance on what information should be provided for each document (such as the FD, SS, etc.). However, such guidance is for the completed finalized documents; the standard encourages production of documentation in step with development. This may result in production of incomplete versions of the documents. The documents would be finalized at the end of the development effort.

Documentation Topics and Their Applicability

F D	S S	P M	U M	O M	
m	m	m	m	m	1. GENERAL INFORMATION.
m	m	m	m	m	1:1. Identification.
m	m	m	m	m	1:2. Purpose of this Document.
m	m	m	m	m	1:3. Validity of this Document.
m	m	m	m	m	1:4. Summary.
					1:5. Background.
x	x	x	x	x	1:6. Related Projects.
x	x	x	x	x	1:7. Definitions, Acronyms, and Abbreviations.
x	x	x	x		1:8. References.
m	m	m	m	m	2. MANAGEMENT.
m	m	m	m	m	2:1. Key Personnel.
x	x	x			2:2. Assumptions, Constraints, and Prerequisites.
x	x	x			2:2.1. User Obligations.
x	x	x			2:2.2. Developer Obligations.
x	x	x			2:2.3. Resource Commitments.
x	x	x			2:2.4. Test Commitments.
x	x	x	x		2:3. Change Control.

Documentation Topics and Their Applicability

F S P U O M
D S S M M M

Only the topic numbers are shown for sections three and four.

m m m m m m 3. SYSTEM-WIDE or GLOBAL INFORMATION.

4. LOCAL, COMPONENT, or SUBSYSTEM INFORMATION.

m m m m m m :1. Overview.
m m m m m :1.1. Functions. [Alternatively: "Components" or "Subsystems"]
x x x x x :1.2. Logic.
x x x x x :1.3. Data Flow.
x x x m m m :1.4. Operation.
x x x m m m :1.4.1. Timing.
m m m :1.4.2. Run Logic.
x x x x x x :1.4.3. Error Conditions, Failure Contingencies.
x x x x x :1.5. Performance.
x x x x :1.5.1. Applicable Standards.
x x x x :1.5.2. Test and Verification Criteria.
x x x x x :1.5.3. Tolerances.
x x x x :1.5.4. Flexibility.

x x x x x x :2. Inputs, Outputs, and Products.
x x x x x :2.1. Data Items
x x x x x :2.2. Data Bases.
x x x x x :2.3. Records.
x x x x x :2.4. Files.
x x x x x x :2.5. Forms.
x x x x x x :2.6. Reports.
x x x x x x :2.7. Publications.
x x x x x x :2.8. Graphics.
:2.9. Other Inputs, Outputs, and Products.

x x x x x :3. Interfaces.
x x x x x :3.1. Languages.
x x x x x :3.2. Commands and Dialogues.
x x x x x :3.3. User Hardware.
x x x x x :3.4. Graphics.
:3.5. Other Interfaces.

m m m m m m :4. Operating Environment.
x x x m m m :4.1. Sites.
x x x x x x :4.2. Hardware
x x x x x :4.3. Support Software.
x x x x x x :4.4. Telecommunications.
m m m m m m :4.5. Security and Privacy.
x x x x x x :4.6. Controls.
:4.7. Additional Requirements.

x x m m :5. Programming Considerations.
m m :5.1. Languages, Operating System, and Support Software.
m m :5.2. Location of the Source and Object Code.
m m :5.3. Program Design Logic.
x x :5.4. Design of Representations

Documentation Topics and Their Applicability

F D	S S	P S	U M	O M	M M	
	x		x			:5.5. Coding Conventions.
	x		x			:5.6. Interaction with Operating System.
x	x	x		x		:5.7. Testing.
x	x	x		x		:5.7.1. Test Plan [alternatively, Plan of Test]
				x		:5.7.2. Results of Tests.
	x		x			:5.8. Storage Requirements.
						:5.9. Additional Discussion.

5. NOTES AND REFERENCES

[1] The opinions in this paper are those of the author and not necessarily those of the Air Force Data Services Center. Coresponderce concerning this paper should be addressed to the author at P.O. Box 7541, Washington, DC 20044.

[2] U.S. Department of Commerce, National Bureau of Standards; "Federal Information Processing Standards (FIPS) Publication 38, Guidelines for Documentation of Computer Programs and Automated Data Systems"; Springfield, VA; National Technical Information Service; February 15, 1976; 55pp.

[3] T. Kurihara, S. T. Redwine, Jr., and S. A. Zaveler, "Observations on Documentation Standards Revision: FIPS Pub 38 after Four Years," IEEE Computer Society, Software Engineering Standards Application Workshop Proceedings, August 18 - 20, 1981, San Francisco, California, 1981, pp 70 -76.

6. ACKNOWLEDGEMENTS

The author appreciates the comments from T. Kurihara, S. T. Redwine, A. J. Neumann, and P. Powell, as well as those from his colleagues at work.

7. CONCLUSION

Modernization of FIPS 38 is needed and is feasible as suggested by this paper.

3. A Proposed Documentation Standard Based on a System Decomposition and Information Base Approach, Saul A. Zaveler.

3.1 Summary of Discussions

There is an analogy between documentation and software. We can learn how to structure documentation from how we structure software.

The Parnas method of applying software structuring principles to documentation was discussed. Parnas supports two principles in particular:

- Information hiding -- if it's likely to change or if no one else needs to know about it, hide it away.
- Separation of concerns -- if items are not similar, don't put them together.

Parnas' software requirements and design documents were listed. The discussant then asked the speaker whether the Parnas approach fit in with the speaker's approach. The speaker replied that the type of standard he is proposing is compatible with Parnas as well as with the many others he had enumerated. The information is similar, but it is organized differently with different terminology.

The second discussant commented that we have heard some very specific considerations relating to the need to revise FIPS Pub. 38. It seems that, implicit in the method of divorcing the display requirements from the information collecting requirements, is the answer to an earlier question on getting usable documentation. Documentation will be more palatable if the producers can update as they go along, without worrying about how it will appear later in the display.

It may be difficult, while defining the requirement, to also define the functions. In fact, functional requirements breakdowns may not be appropriate, especially for higher levels of documentation. The current standard doesn't tell what to document first, only what is left out.

3.2 Summary of Questions

- Q: There are many overlapping, even contradictory standards, within the Government. Can these be coordinated?

  A: One way to resolve the severity of major conflicts is to define requirements to be met by the standard until they become very clear, then the solution becomes apparent. To solve the problem of coordination, develop a reference model first, then develop the standards in subparts that will fit in the total picture.

- Q: What about standards for large, decentralized systems?

  A: The guidelines and standards for small systems are a subset of the large. If you create a set of standards that encompasses the larger system, the smaller system will have less complex requirements. Develop subset standards of big systems so they represent the small and medium, then develop samples for each (small, medium and large) to use as teaching tools.

- Q: Does it pay to document a small system?

  A: Most organizations have a threshhold of time and money below which documentation is not required. This is based on the assumption that the software can be rewritten with less cost. However, lack of documentation makes data unaccessible, and this approach may be a self-fulfilling prophecy: if you don't document, you will have to do it over the next time.

MODERATOR'S COMMENTS

Based on the papers presented, questions raised, answers and discussions, the following conclusions/recommendations were reached during this session:

1. Why do we need documentation?

   To know what the documented product does, how, and for whom. Without this information, the product (software, procedures, system, etc.) may be lost. The level of documentation must be cost effective. The user of the product must be able to understand and use the product easily. If there is management involved (usually, except for user-owned microcomputers), the documentation must allow management information for planning, accounting, control, etc.

2. What should be the process of developing better documentation standards?

   To develop such standards, the systems (project) approach should be used throughout the process. Emphasis should be placed on coordination, overall applicability, and consensus.

3. What should be the characteristics of better documentation standards? (See the first paper for more details.)

   Documentation standards must be stated in a conceptual project context; they must be adaptable to each project/system's situation; they must be complete and thorough with ample definitions and guidance for the authors; they must be manageable and based on what is known to work. They must also assist the authors in providing documentation that is organized, logical, objective, tailored to the reader/user, clear, concise, and that contains all of the information needed and nothing else.

4. How should documentation standards assist in the production of documentation?

   They should provide guidance for context, concepts, terms, content, and format. They also should encourage the use of state-of-the-art approaches and tools available for documentation. (See Session C for more details.)

5. Who should produce documentation?

   The standards should be written for technical writers, who should produce the final documentation and who may or may not have any experience in computers, programming, systems development, implementation and operation.

Microcomputer Systems Users Need Better Documentation

Richard A. Bassler, PhD, CDP, CDE

Professor of Computer Systems Applications
Center for Technology and Administration
College of Public Affairs
The American University
Washington, DC 20016

From the initial pioneering days of microcomputers, documentation has been the weak link between use of the machines and the vast number of potential users. Many of the microcomputer hardware and software suppliers of the early days are now sufficiently large and financially healthy enough to be able to spend resources toward communication with users. The success of the microcomputer has attracted large computer manufacturers to the marketplace. Large manufacturers such as IBM and Xerox are likely to make sure that the user will have documentation that is usable. With the proliferation of microcomputers, user's groups have created a form of documentation that is verbal. This mutual aid made microcomputing survive, if not flourish, during its infancy days of less than adequate documentation. Third parties such as publishers have moved in to fill the void in documentation. Rapid expansion and survival within the micro industry may well depend on the quality of the documentation furnished.

Keywords: beginning computer users; documentation; hardware systems documentation; large computer manufacturers; microcomputers; periodical literature and documentation; software documentation; user's groups; verbal documentation.

1. INTRODUCTION

Microcomputers have a unique place in the hierarchy of computing. Arriving on the scene only recently, they have opened up a totally new marketplace for the industry. Along with this new marketplace comes a new class of computer users. These new users are not sophisticated computerists. It is more likely that they do not ever want to become really knowledgeable about computers and computing. They want to **use** the cheap computer power of this technology for their applications instead of learning the technology.

1.1 The Early Days

The first microcomputers were typically in kit form. Most popular of these were the Altair series of computers, a combination which really was the prime mover in getting microcomputers into the hands of hobbyists. Potential users would buy a collection of parts, usually unidentified in plastic bags, and a few sheets of documentation as assembly instructions. Early pioneers were a brave lot, and most of them were ham radio operators or other folk who had some experience with either hardware or software. The next stage in this evolution was the arrival of more sophisticated kits with some software, such as the Processor Technology series of SOL computers and the IMSAI. These microcomputer systems eventually were delivered in assembled form. As a result, the number of nontechnically oriented users began to increase rapidly.

Documentation for these systems began to show improvement. The manual for the SOL 20 (1) was an example of attempts to produce adequate documentation. Although it was still oriented toward the kit builder, it did have sufficient information in it to make it tolerably useful to the sophisticated user. The manual was not indexed, and the user was at the mercy of using a sequential search to find any particular item.

1.2 Current Systems

Recent microcomputers, such as the OSBORNE 1, have user's manuals directed toward the user with little experience. For example, the **OSBORNE 1 User's Reference Guide** (2) states in its introduction that:

> The OSBORNE 1 has been designed to make you more productive in your professional life. If you are a typical OSBORNE 1 owner, you will probably never write a computer program; you do not have to in order to use this computer. In fact, you can discard the traditional concept of computers. Your OSBORNE 1 will perform a multitude of tasks for you. You simply instruct the computer by having it play a program on a diskette, much as you would instruct your stereo system by playing a record or cassette tape. In fact, learning to use your OSBORNE 1 computer is no more complex than learning to use a stereo system.

This is an indicator of the marketplace that the current crop of microcomputer manufacturers is aiming at. IBM, Xerox, Wang, and most other large vendors have also fielded small computers aimed at this burgeoning new arena for the sale of large volumes of computer systems. It turns out that the process of using **any** computer is a little more complex than Osborne suggests. If the whole movement toward mass use of microcomputers is to be truly successful, a great deal of attention needs to be paid to the documentation of both hardware and software.

2. HARDWARE SYSTEMS DOCUMENTATION

2.1 Trend toward Improvement

Many of the hardware manufacturers who cater to the microcomputer market have been paying attention to the needs of their potential customers. Well, almost. There is still a long way to go before the documentation for hardware will be really useful to the users who do not want to be technicians.

One example of such improvement is in the documentation for the Diablo 1650 word processor printer. This paper was printed on such a printer/terminal driven by a state-of-the-art microcomputer. The author has struggled with these two manuals for over a year. The documentation for this printer, available in 1979, is in two books. The first of these, the **Product Description** (3), and the other, the **Maintenance Manual** (4), were written for the technically oriented person. The casual user could not fathom what was included and could not begin to use the devices from these manuals. They are useful to technicians, but only to those familiar with the manuals. The lack of an index inhibits really good usage.

Later Diablo machines were delivered with an operator's guide (5) that was written for the technically unsophisticated user. It describes how to use the external features of the machine in terms that most WP operators can fathom. Although this was a step forward, the newer operator's guide would be more usable with an index. Of course, in its 25 pages, one could not find the detail that was in the 500 pages of the earlier manuals. The complete manuals are still available for those who want to get into the details and advanced features of the printers.

2.2 Hope for the Future

With the arrival of the IBM Personal Computer (6) and the Xerox 820 (7) computer, it is hoped that the large computer manufacturers will apply their experience in dealing with many levels of consumers toward developing documentation to make their small computer systems usable by a wide variety of users. Failure to do so will result in slowing the forward movement toward the exploitation of this potentially huge marketplace.

The idea that these computers will be sold through unusual (for computers) marketing schemes, such as IBM in Sears Roebuck stores, means that the instructions for use must be clear and understandable. For those machines sold through computer stores such as Computerland, more help would be available from the store staff to aid the buyer.

3. SOFTWARE SYSTEMS DOCUMENTATION

3.1 The Early Days

An early effort in large-volume software documentation consisted of six manuals that described the features of the CP/M operating system. CP/M is today the preeminent microcomputer operating system with almost all microcomputer manufacturers offering it as their standard operating system or as an option. Where CP/M is not furnished by the vendor, it is sometimes available from a third party software supplier. The original six manuals (8) could be kindly described as virtually unreadable by the less experienced. Jargon of the microcomputerists abounded. Terms were undefined in the document or for that matter anywhere else. The final blow was that no index to these documents was furnished.

When a user has considerable help from other users, it is possible to understand what the manuals are describing. This form of tutorial by other microcomputer enthusiasts was the true form of documentation of the early systems. This method of communication often exists today. There is really no substitute for the "hand holding" method of learning to use a microcomputer system.

3.2 Mutual Aid and Verbal Documentation

The microcomputer user's group has emerged as a viable way for an individual to understand the use and peculiarities of any system. In fact, it might be viewed as a forum of verbal documentation. Experience is passed on to the newcomers entering a group of microcomputer users. Every city has such groups. They are usually formed around a background of some common denominator. In the Greater Washington, DC area, there are user's groups built around the CP/M operating system and the Apple, Radio Shack TRS80, and Osborne microcomputers. Nationally, the CP/M User's Group distributes a wide variety of public domain software in magnetic form. Many of the local user's groups perform a secondary distribution service by making several thousands of programs available.

Help in using these software packages is needed. The documentation, if any exists, is usually poor. By seeking help from others who have implemented such packages, one may be able to make use of the programs. Also, there is a learning experience connected with this sharing that is invaluable.

3.3 Software Vendors' Attention to Documentation

In recent years, there has been a noticeable improvement in the documentation of application software for micros. Some of the larger vendors have attempted to produce manuals and user's guides that are readable and understandable by a typical user. For example, a reference manual for the Radio Shack TRSDOS (Disk operating system) (9) is rather good. It is a carefully typeset, printed manual, with attractive illustrations and enough white space to make it readable. The manual begins with an elementary overview of the hardware connected with this particular operating system. Each feature of the OS is explained in careful detail, and it would be expected that most users would understand what is being explained. There is even a three-page index to the manual. This will help in finding reference items. Documentation for other systems such as the Apple and the Heathkit computers has improved.

3.4 Publishers' Attempt to Fill Void

With all the problems about the CP/M operating system documentation, it is surprising how long it took for the publishing industry to come to the rescue. At this writing there are at least

four 1981 books explaining how to get the most out of the CP/M operating system. Typical of these, and probably the easiest to use and most comprehensive, is Thom Hogan's book (10) on how to use CP/M. This is clearly written and has an elementary introduction for the beginner. It explains much of what is covered in the original CP/M manuals. Even experienced users are learning from this book and reaching an understanding of what they have been doing.

Besides covering operating systems, publishers are helping to fill the void in documentation for several computer languages. Several books about Microsoft BASIC are available. CBASIC2 is the subject of a volume (11) devoted to more extensive examples about the syntax and use of the language than were furnished by the vendor.

4. SOFTWARE FROM THE PUBLIC DOMAIN AND THE PERIODICAL LITERATURE

4.1 User's Group Software Documentation

When software is passed on through informal user groups, typically the documentation is passed on in a simple manner. It will probably consist of user instructions on how to run the program. If there is some user interaction to the program, it may be possible to run the program without help. Frequently, the source code, if furnished, will contain enough remarks to make it possible to fathom. This is fine for the experienced user, but the beginner is usually lost without some help from more detailed user documentation.

All software that is shared by user's groups is really not in the public domain. Much of it is pirated commercial software. The documentation for these programs frequently is a multi-generation xerographic copy that may or may not be readable. Part of the blame for the extensive piracy of software, passing in the name of "sharing," is caused by the inability of the potential users to really know whether a program will run on their system and whether it will do the tasks expected of it. Some users of this class of software rationalize their pirating with the explanation that they are "testing for evaluation" in their user environment. Some intend to, and do, buy legitimate copies when proven useful in their own system and application.

4.2 Programs in the Periodical Literature

All the journals that have sprung up in the microcomputer area in the past few years are publishing programs of virtually any type. Some are well documented, others are not. Editorially, these journals have a long way to go to achieve the quality desired. Frequently programs are published that simply will not run as listed in the journal. Later editions of that journal will have corrections listed in the letters to the editor. Of course, one can write off to an educational foray the experience in trying to get some of this software to work.

Some of the code written for these journals is written in the style of a few years ago, when micro memory was expensive. Today, when 64K memories are the norm for serious users, BASIC programs written in a style to save on memory usage are unacceptable. Writing clearly documented programs that are readable to the user is more important than trying to save bytes in internal memory usage. Progress is being made along these lines.

5. DOCUMENTATION STANDARDS FOR MICROCOMPUTER SYSTEMS

5.1 Consumerism in the Software Marketplace

The best way to get better documentation for microcomputers is through consumer action. When potential buyers have good evaluations of available software to pass judgment on the quality of the vendor-supplied documentation, then resistance in the marketplace would take care of the poor cases. The periodicals are beginning to pay attention to the quality of vendor documentation in their reviews of both hardware and software. **InfoWorld** regularly reviews both hardware and software that is available in the marketplace. Documentation is one of the five criteria for

evaluating a software package. The ratings assigned are poor, fair, good, and excellent. A recent review (12) involved a Hewlett-Packard hardware and software package called **Business Assistant.** The documentation for this system was rated excellent. The reviewer comments:

> Hewlett-Packard has resisted the temptation to which most CP/M compatible hardware suppliers have succumbed. The HP-125 comes with a totally rewritten and indexed manual for CP/M. Gone are the traditional set of seven Digital Research manuals that you must read in parallel to comprehend them. This feature alone is nearly worth the price differential between the 125 and less expensive systems.
>
> The reference manual is well written and explains exactly how things work in the HP-125 implementation of CP/M rather than employing the traditional approach of supplying a few scant notes and the Digital Research CP/M Alteration Guide. (12)

In a buyer's market, the consumer is a powerful force. With the low price of microcomputer systems and their related software, the only way to survival is through volume. No longer can a software or hardware vendor expect to capture the development cost of a software package in the first few sales. Competition is fierce in the microcomputer field. The user with a CP/M-based system can select from a dozen excellent word processing packages. The one that furnishes the best documented user instructions and that can accomplish all the tasks of the application will win. The user can easily evaluate the quality of the documentation. Manuals for virtually every micro software package can be purchased separately from the software for evaluation purposes.

5.2 Need for Experts in Communicating

Computer documentation in general, and for the microcomputer in particular, is too important to be left to the computer expert. What is needed is a whole new generation of experts on communicating technical ideas in the language of the user. We can no longer expect that the user of computers will learn the language of the technology. There are many potential users who have no computer background and no desire to attain this expertise. The documents furnished with micro software will be the determining factor in the consumer acceptance of any software package.

5.3 Importance of Index

Few, if any, microcomputer manuals have been indexed until recently. Reviewers have complained about the absence of indexes, or of indexes of poor quality. Virtually every book in the author's library is indexed except for many of the micro applications manuals. The task is not difficult if done by someone with experience. Having a workable index or not in a manual may make the difference between survival and disaster for a micro vendor.

5.4 Testing the Documentation

The documentation must be tested in the user arena. No longer will it be acceptable to pass it along in the office to see if all the technical jargon is correct. Tests must be made with the users who will have to cope with the foibles of the documentation. They must be allowed to pass along comments and suggestions that must be listened to and incorporated in the documentation before the final versions are published. Frequent revisions must be made to correct the problems discovered though use. The cost of this must be part of doing business in this high-volume arena.

CONCLUSION

Early microcomputer systems were notable for the poor quality and quantity of their documentation. The industry has recently begun to recognize the need for quality documentation oriented toward the mass market of microcomputer buyers. Possibly the survival of the microcomputer industry, and certainly its growth, will be affected by the recognition of this need. Quality documentation can come only from putting communication experts on the task. Hardware and software advances were brought about by putting design and engineering experts to work on the task. The future looks bright and exciting.

7. REFERENCES

1. **Sol Systems Manual,** 6200 Hollis Street, Emeryville, CA 94608, Processor Technology Corporation, 1976, 700+ pages (pages numbered within chapters), not indexed.

2. **OSBORNE 1 User's Reference Guide,** 26500 Corporate Avenue, Hayward, CA 94545, Osborne Computer Corporation, Undated (presumably 1981), 225+ pages (pages numbered within chapters), not indexed, table of contents without page numbers.

3. **Series 1650 Word Processor Printers and Terminals, Product Description,** Preliminary, Document Number 90402-00, 3190 Corporate Place, Hayward, CA 94545, Diablo Systems, Inc., March 1979, 200+ pages (pages numbered within chapters), not indexed.

4. **Series 1640/1650 Printers and Terminals, Maintenance Manual,** Preliminary, Document Number 90413-00, 3190 Corporate Place, Hayward, CA 94545, Diablo Systems, Inc., July 1979, 300+ pages (pages numbered within chapters), not indexed.

5. **Series 1650 Word Processor Terminal Operators Guide,** Document Number 90405-02, 24500 Industrial Boulevard, Hayward, CA 94545, Diablo Systems Incorporated, July 1980, 25 pages, not indexed.

6. Lemmons, Phil. "The IBM Personal Computer: First Impressions," **BYTE,** Volume 6, Number 10, pp. 26, 28, 30, 32, 34, October 1981.

7. Maggie Cannon. "Major Manufacturers Eye World Market for Small Computers," **InfoWorld,** Volume 3, Number 20, p. 51, October 5, 1981.

8. **Digital Research CP/M Documentation Version 1.4** (six volumes):
   I. **An Introduction to CP/M Features and Facilities,** 35 pages;
   II. **CP/M Assembler (ASM) User's Guide,** 22 pages;
   III. **CP/M Dynamic Debugging Tool (DDT): User's Guide,** 19 pages;
   IV. **CP/M System Alteration Guide,** 23 pages + appendices;
   V. **CP/M Interface Guide,** 38 pages;
   VI. **ED: A Context Editor for the CP/M Disk System: User's Manual, 17 pages,**
Box 579, Pacific Grove, CA 93950, Digital Research, 1976, not indexed.

9. **TRSDOS & Disk Basic Reference Manual,** One Tandy Center, Fort Worth, TX 76102, Radio Shack, a division of the Tandy Corporation, 1979, 200+ pages (pages numbered within chapters), indexed.

10. Hogan, Thom. **Osborne CP/M User Guide,** 630 Bancroft Way, Berkeley, CA 94710, OSBORNE/McGraw-Hill, 1981, 283 pages, indexed.

11. Osborne, Adam, Gordon Eubanks Jr., and Martin McNuff. **CBASIC User Guide,** 630 Bancroft Way, Berkeley, CA 94710, OSBORNE/McGraw-Hill, 1981, 215 pages, indexed.

12. Milewski, Richard A. "The HP-125 Business Assistant from Hewlett-Packard," InfoWorld, **Volume 3, Number 20, pp. 37-39, October 5**
1981.

2. Microcomputer Systems Users Need Better Documentation, Richard A. Bassler

2.1 Summary of Discussions

The problems of documentation for microcomputers are an illustration of "we learn from our mistakes -- we have learned how to do these mistakes well." If we knew how to produce good manuals for minicomputers and mainframes, we would know how to produce good ones for micros. Since micros are bought by the user, the microcomputer market may assume a leading role in forcing us to do a better job of documentation.

A tongue-in-cheek comment was made that the manuals are the real thing. The hardware is merely an attempt to meet the manuals. A number of extensive disclaimers were read to support this view.

2.2 Summary of Questions

- Q: How can we incorporate standards for micros into the classic type of developmental systems? Are they two separate problems?

  A: Yes. This documentation is for naive users.

- Q: Do you have any proposed standards for documentation of microcomputer application programs and systems?

  A: Documentation must be approached each time on a unique basis. It must be comprehensible and understandable to the naive user.

INTRODUCTION

Lenore S. Maruyama, Moderator

Network Development Office, Library of Congress
Washington, D.C.

This session considers different aspects of software sharing which, in a broad context, encompasses the use of software created by persons or organizations outside one's immediate institutional affiliations. Because of the increasing importance of software sharing in all sectors (public/private, for-profit/not-for-profit), standards and standardized techniques have been or are being developed to facilitate the sharing process.

In a recent report, An Assessment and Forecast of ADP in the Federal Government (published by the National Bureau of Standards based on research conducted by International Data Corporation), it was estimated that the Federal government spent about $559 million on software in 1979. (1) These expenditures were categorized as follows:

$117 million (21%)	Systems Software
66 million (12%)	Utility Software
376 million (66%)	Applications

Of the $117 million spent for systems software, about 53% ($62 million) had been spent on internally developed software; of the $66 million for utility software, 50% ($33 million) had been spent on internally developed software; and of the $376 million for applications, 83% ($312 million) had been spent on internally developed software. These figures are important not only in terms of the standards and standardized techniques being discussed in this session but also for the entire software sharing process which has the potential of saving the Federal government (and others) substantial amounts of money.

The first three papers address various information-gathering techniques. "Effective Bibliographic Standards for Computer Software: Improved Documentation and the Need for 'Title Page' Equivalents" by Sue A. Dodd presents bibliographic procedures on how to identify and describe a computer program and proposes the inclusion of a user header label to record the necessary bibliographic information. Ms. Dodd is an associate research librarian in the Social Science Data Library at the University of North Carolina, and following several years of active involvement in the area of bibliographic control for machine-readable data files, she has compiled a cataloging manual for machine-readable data files. "Standards for Bibliographic Control of Machine-Readable Data Files" by Lenore S. Maruyama describes standards in the area of automated bibliographic control, including a new format for machine-readable data files, and reviews the elements in FIPS 30 for software summaries as well as proposes a new standard for numeric data files. Ms. Maruyama is a senior information systems specialist in the Network Development Office of the Library of Congress, and in addition to her involvement with numerous automation projects at the Library, she has compiled seven MARC (machine-readable cataloging) formats, the most recent being the one for machine-readable data files. In "The Computer Program Abstract as Software Documentation," Margaret K. Butler discusses the development of the recently completed American National Standard for Computer Program Abstracts and its role in software sharing. Mrs. Butler is a senior computer scientist at Argonne National Laboratory and director of the National Energy Software Center. She has participated in standards work as a member of the ANSI X3 SPARC (Standards Planning and Requirements Committee) and chaired Technical Committee X3K7 on Computer Program Abstracts.

The remaining three papers present techniques that are intended to facilitate the sharing process. "An Integrated Machine-Readable Data Documentation System" by Richard C. Roistacher describes the use of a text formatter to manipulate machine-readable text, including the preparation of documentation. Mr. Roistacher is a research associate at the Bureau of Social Science Research in Washington, D.C., and his major research interests are in the areas of social science computing and data analysis and the social and organizational effect of computer networks. (Although Mr. Roistacher's paper has been included in these proceedings, he was not able to present the paper at the workshop.) In "Compilation of Bibliographic Data Element Dictionaries," Madeline M. Henderson describes a project in progress to standardize techniques to compile a data element dictionary for all elements used in processing Federal documents. Mrs. Henderson, now a consultant in information management, analysis, and assessment, had been manager of the ADP Information Analysis Project in the Institute for Computer Sciences and Technology of the National Bureau of Standards until her retirement in 1979. Finally, "Capital Games: the Problem of Compatibility of Bibliographic Citations in Data Bases and in Printed Publications" by Hans H. Wellisch discusses the problems of converting data from bibliographic files to references or footnotes in monographs or journals because of differing capitalization practices in the latter. Mr. Wellisch is a professor in the College of Library and Information Services at the University of Maryland, and his major interests include subject indexing, the linguistic aspects of information science, the physical planning of libraries, and the history of library and information work.

REFERENCES

(1) Gray, Martha Mulford. An Assessment and Forecast of ADP in the Federal Government. Washington, Institute for Computer Sciences and Technology, National Bureau of Standards, 1981. p. ix, x. The author explains that these figures were derived by analyzing software spending patterns for the U.S. general purpose computer population and, by extrapolation, estimated for the Federal government.

Effective Bibliographic Standards for Computer Software: Improved Documentation and the Need for "Title Page" Equivalents

Sue A. Dodd

Institute for Research in Social Science
University of North Carolina

Bibliographic control over computerized information has slowly been evolving within the library and information science profession during the last decade. A major landmark that helped to focus increased interest in bibliographic control of computerized information was the inclusion of Chapter 9 on machine-readable data files (MRDF) in the second edition of the Anglo-American Cataloguing Rules (AACR2). Publication of these rules in 1978, coupled with a number of other events, including the compilation of a MARC (MAchine Readable Catalog) format for MRDF provided some important tools for establishing bibliographic control over the proliferation of data files and computer software. General purpose computer software must be properly identified with sufficient bibliographic data elements to be processed in turn by librarians and information scientists, converted into catalog records and data abstracts, and finally integrated into existing automated retrieval systems. This paper presents procedures on how to identify and describe a computer program and is directed at those government producers who have the responsibility for providing descriptive information on available Federal software, and for seeing that such information reaches its intended audience.

Bibliographic control; Bibliographic standards; Computer software; Documentation standards; Machine-Readable Data Files (MRDF)

1. INTRODUCTION

The Federal Software Exchange Program was established in 1976 by General Services Administration (GSA) to promote Government-wide sharing of common use software owned by Federal agencies. In order to provide for a centralized unit for information on available software, GSA established the Federal Software Exchange Center (FSEC) at NTIS. The vechicle for Federal software information and dissemination is the Federal Software Exchange Center Catalog. It contains software abstracts covering numerous application areas on programs that are written in a broad range of computer languages for a wide variety of hardware. The source of information for the abstracts is the Federal Information Processing Standard Software Summary (FIPS Pub. 30). According to Government regulations (FPMR 101-36.1068 and 101-36.169), before a Federal agency can obtain a Delegation of Procurement Authority (DPA) to acquire software from commercial sources, it must screen available Federal ADP resources by reviewing the Federal Software Exchange Center Catalog. Even with these regulations and information gathering procedures, it is commonly observed that available government-produced software programs are seriously underutilized both inside and outside of government. This condition is most often attributed to the lack of adequate information about and access to available software. Notable symptoms of the problem include:

-- confusion and exasperation in trying to locate and acquire computer programs within the Federal government
-- insufficient information on the most recent version or editions of government-produced software
-- published catalogs quickly becoming out-of-date with long time periods between updates or new releases of catalogs
-- no standardization across agencies except for the Federal Information Processing Standards (FIPS) of the National Bureau of Standards which are often "observed only in the breach" -- in addition, such forms are often completed by persons not directly involved with the development of the program being described
-- waste in government spending due to a duplication of effort; new software programs are being budgeted and developed when similar programs already exist within the Federal government

While all of these problems cannot be solved by a single solution, many can be alleviated by adhering to the recently formulated bibliographic standards for machine-readable data files (MRDF), [1] which includes computer software; by integrating these standards into existing documentation; and by converting this information into a centralized automated online retrieval system. The process of developing an efficient information retrieval system must include the development and enforcement of standards governing bibliographic descriptions of computer programs and their accompanying documentation. Bibliographic control over computerized information has slowly been evolving within the library and information science profession during the last decade. A major landmark that helped to focus increased interest in bibliographic control of computerized information was the inclusion of Chapter 9 on MRDF in the second edition of the Anglo-American Cataloguing Rules (AACR2). Publication of these rules in 1978, coupled with a number of other events, including the compilation of a MARC (MAchine Readable Catalog) format for MRDF provided some important tools for establishing bibliographic control over the proliferation of data files and computer software. Not unlike an author who is asked to follow standard descriptive rules for identifying his unique creation, and not unlike publishers who have established among themselves standard publishing practices for identifying their publications, government agencies should agree to follow accepted bibliographic practices for describing and documenting computer software. General purpose computer software must be properly identified with sufficient bibliographic data elements to be processed, in turn, by librarians and information scientists, converted into catalog records and data abstracts, and finally integrated into existing automated retrieval systems. This paper presents procedures on how to identify and describe a computer program and is directed at those government producers who have the responsibility for providing descriptive information on available Federal software, and for seeing that such information reaches its intended audience. The individual descriptive data elements required to establish bibliographic identity for data files and programs are outlined. Immediately following are instructions on how to prepare "title page" equivalents (both internal and external).

2. DATA ELEMENTS TO IDENTIFY SOFTWARE

Conceptually, data elements for computer software can be broken down into at least six levels: those needed to identify a program (e.g. bibliographic elements); those needed to describe the contents of a program (e.g. descriptive summary or abstract); those needed to classify a program (e.g. appropriate classification codes, indexing, or subject descriptors); those needed to access or use a program (e.g. physical and technical characteristics, computer compatibility, peripheral requirements); those needed to analyze or operate a program (e.g. citation of documentation, related publications); and those needed to archive or maintain a program (e.g. agency records pertaining to the development, use, and storage of the program). Only the selected data elements necessary to identify a computer program will be discussed here, but the reader is referred to other publications [2] dealing with other descriptive levels including how to compile an abstract.

Title. A formal computer software title should be distinct from a data set name or other computer-related working names. Acronyms should be avoided, but if used, they should be explained in the secondary title or subtitle. For example:

TRMETS: Trace Metals Analyses

With existing technology for keyword or full-text retrieval, any descriptive words contained in the title take on added significance. Consequently, a titles for software should be descriptive of the major functions or contents they are describing.

Authorship. Give the full name of author(s), programmer(s), or corporate body (government agency) responsible for the intellectual content of the software. For multiple authors, give proper name order.

Statement(s) of responsibility. Statements of responsibility relate to persons or corporate bodies responsible for intellectual content; to corporate bodies from which the content emanates; to personal or corporate sponsorship; and to personal or corporate responsibility for the performance or development of a work. For books, statements of responsibility include authors, editors, writers of prefaces or introductions, illustrations, etc. For computer programs, statements of responsibility are used for authorship, but also for contributing roles played in the creation or development of the program. Since many of the tasks associated with program development may be contracted out to another party, responsibility statements may be used to indicate the relationship of the work to persons or corporate bodies that would otherwise not be known. Example:

> Developed by the Center for Advanced Computation, University of Illinois for the Office of Land Information and Analysis, National Aeronautics and Space Administration (NASA)

Edition. A program edition occurs when there are major changes in the programming statements, a change in the programming language, or other significant changes. Editions for programs are usually designated in terms of "version," "release" and "level." Each signifies to a potential user that new routines, or other enhancements have been made to the program. Examples:

Version 5.20 or Level 3.4

In both examples above, the term "release" is implied but not stated. If it were stated, it would be "version 5, release 20" or "level 3, release 4."

Producer. A software producer is defined as that person or corporate body with the overall administrative responsibility for bringing the program into existence.

Place of production. Give the complete address (as judged appropriate) for the place of production. The place of production is defined as the "main office" or the location used as part of the formal mailing address. For example, the place of production for a program produced at the Bureau of the Census is:

Washington, D.C.

not

Suitland, Md.

Name of personal or corporate producer. Give the personal or corporate producer of the program. In citing the appropriate organization or person responsible for the production of the program, indicate the full organizational title and affiliation as appropriate.

Date of production. Production date for a software program is defined as the date the file became operational in machine-readable form for analysis and processing.

Distributor. A program distributor is defined as that organization which has been designated by the author or producer to reproduce copies of a particular program. If a distributor is not cited, it is assumed that the author or producer is fulfilling this function.

Place of distribution. Give the complete geographic location of the place of distribution, including address, zip code and telephone number.

Name of personal or corporate distribution. Give the personal or corporate name of the distributor of the program.

Date of distribution. The distribution date for a program is defined as the year the program became available for wide distribution to the public or to other agencies. If the distribution date is not different from the production date, then it is not given.

Series title. A program series title is the collective title under which a program is issued as one of its parts. Like a monographic series for printed material, the program file conterpart may be defined as a number of separate program files issued successively and related in subject, purpose, and form. They are usually produced by the same organization, but may have separate and distinct titles. Such program-defined series should carry a collective title and be placed on the title page of the accompanying documentation.

3. "TITLE PAGE" EQUIVALENTS FOR SOFTWARE

Internal title page. Optimally, a "title page" equivalent should be part of the program itself. An internal title page would be totally file-specific and could not easily be lost or separated from the program it is describing. Upon receiving a program stored on any type of data carrier, the recipient could arrange for the "title page" equivalent to be printed out or displayed on a video/CRT screen. The resulting descriptive information would reveal the program's bibliographic identity and any relevant information pertaining to its edition.

Internal user file labels. The ANSI X3.27-1978 - Magnetic Tape Labels and File Structure for Information Interchange provides magnetically recorded labels for the purpose of identifying files. Although limited to magnetic standard-labeled tape files, an option in the standard establishes a User Header Label (UHL) in which space adequate for a "title page" equivalent for computer software is available.

A UHL consists of the following fields of eighty character records:

CP	Field Name	L	Content
1 to 3	Label identifier	3	UHL
4	Label Number	1	"a" character
5 to 80	Reserved for User Application	76	"a" characters

Legend
CP - character position in the label. Field name - reference name of the field. L - length of the field (number of characters). Content - content of the field as appropriate. "a" - An "a" character is any one of the set of digits 0, 1, 2, . . . 9, the uppercase letter A, B, C, . . . Z, and the following special characters: SP ! " % ' ( ) * + , - . / : ; = ? <>

A completed set of UHLs representing a "title page" equivalent for a software program is given below:

UHL 1 SYMAP: Synagraphic Mapping System. Conceived
UHL 2 and Developed by Howard T. Fisher, Technological
UHL 3 Institute, Northwestern University. version 5.20
UHL 4 Produced by the Laboratory for Computer Graphics
UHL 5 and Spatial Analysis, Harvard University, Cambridge,
UHL 6 Mass., 1975.

User applied information - characters 5-80

Label number - 1 character - position 3

Label identifier - 3 characters - position 1-3

Such information once recorded should be copied without change when the program is recorded on another tape. Information thus supplied should apply to the identity of the file and not to its physical characteristics. Once supplied by a program producer, the information should not be tampered with -- like the title page of a book, it should remain constant.

From the point of view of bibliographic control of computer software, the use of an internal user label or its equivalent is essential. Motion pictures have title and production credits incorporated at the begining of the picture frames, and sound recordings have permanent external labels providing bibliographic identification. However, at this stage of their development, computer software lack any internal/external information or "title page" equivalent sufficient to identify the distinguishing features of specific titles, editions, and production credits.

One problem surrounding the use of internal labels is compatibility. Different computers use different labeling devices that are often not compatible. How problematic such labels will prove to be when considering the tranfer of a program from one computer system to another is difficult to determine. There is also the question of how easily such a label could be bypassed by a system which does not read labels at all or only a particular type of label. Thus the intent of internal user labels to record the equivalent of a title page on the file itself is a useful goal, but it is unclear how quickly such a system of documentation will take hold. It is hoped that appropriate parties such as the National Bureau of Standards or the American National Standards Institute will review this problem and make relevant recommendations. Perhaps this workshop could serve as the initial forum on this problem.

External title page. The alternative to an internal "title page" equivalent, is to rely on the title page of the accompanying documentation to adequately describe the associated program. However, program producers must keep in mind that title pages compiled as part of the program's accompanying documentation must serve a dual function -- that is, the information must be sufficient to identify the printed documentation, the program being described, and the institutional origins of both. Unlike the program it is describing, documentation can be functionally independent. A program on the other hand, connot be implemented without the aid of some type of documentation -- and the linkage between the two must be maintained. One current practice that provides this linkage is an introductory statement at the top of the documentation's title page indicating that the work being described is a user's guide or manual to be used in conjunction with the program and that the information on the title page pertains both to the computer software and its documentation.

Edition statements for the program must be distinguished from edition statements pertaining to the program's documentation. The program file's edition statement precedes the producer statement of the title page, while the edition statement pertaining to the program's documentation follows the program's producer statement, normally at the bottom of the title page (figure 1) [3].

4. CONCLUSION

What does it mean for the user when there are no effective bibliographic standards for documenting computer software? A prospective user of government-produced software faces many problems related to identifying existing program resources. These problems exist in part because bibliographic standards for computer software are not yet enforced. Bibliographic control consists of standards and consistent methods for listing titles, editions, and for naming organizations responsible for producing them. Without these standard procedures, no coordinated system of information services can be created for computer software. Uniform procedures should be developed for the entire Federal System, and these procedures should reflect existing bibliographic standards both for programs and their documentation. Agencies must devote special attention to the problems of adequate bibliographic descriptions and "title page" equivalents for those general purpose computer programs that they have a mandate to share with other agencies.

REFERENCES

1. American Library Association. Anglo-American Cataloguing Rules. 2nd ed. Chicago: American Library Association, 1978, 203 pp.

   Dodd, Sue A. Cataloging Machine-Readable Data Files: An Interpretive Manual (mimeographed). Chapel Hill, N.C.: Institute for Research in Social Science, University of North Carolina, 1981, 396 pp.

   Library of Congress. Machine-Readable Data Files: A MARC Format. Final Draft. Washington, D.C.: Network Development Office, Library of Congress, June 1981.

2. Dodd, Sue A., op. cit.

   Roistacher, Richard C. A Style Manual For Machine-Readable Data Files and Their Documentation. Washington, D.C.: Bureau of Justice Statistics, U.S. Department of Justice, June 1980. Report No. SD-T-3, NCJ-62766. For sale by the Superintendent of Documents, U.S. Government Printing Office.

   United States. Office of Federal Statistical Policy and Standards, Technical Paper No. 3: Procedures for Preparation of Abstracts of Public Use Statistical Machine-Readable Data Files. Washington, D.C.; Office of Federal Statistical Policy and Standards, Department of Commerce (Draft), December 1980.

3. This figure of a proposed title page for computer software was compiled by the author, and does not reflect the actual title page of the SYMAP manual produced by Laboratory for Computer Graphics and Spatial Analysis at Harvard University.

(User's Guide for Computer Software)

SYMAP: Synagraphic Mapping System

Conceived and Developed

by

Howard T. Fisher
Technological Institute - Northwestern University

version 5.20

Produced and Distributed

by

Laboratory for Computer Graphics and Spatial Analysis
Graduate School of Design - Harvard University

1975

User's Guide 5th edition
Revised, October 1975
Revised, May 1976
Revised, September 1979

User's Guide Prepared by James D. Dougenik, Applications Programmer and David E. Sheehan, Computer Cartographer

Harvard University 520 Gund Hall 48 Quincy Street Cambridge, Mass. 02138

Tel: (617) 495-2526

Figure 1: Suggested Title Page for Computer Software Documentation

Standards for Bibliographic Control of Machine-Readable Data Files

Lenore S. Maruyama

Network Development Office, Library of Congress
Washington, D.C.

As a prelude to a review of certain standards in the FIPS series, standards in the area of automated bibliographic control are described, the most important being the American National Standard for Bibliographic Information Interchange on Magnetic Tape (ANSI Z39.2). This standard format structure for machine-readable bibliographic data has been implemented by different bibliographic applications, and the implementation by the library community, exemplified by the series of formats known as MARC (Machine-Readable Cataloging) that were developed by the Library of Congress, is discussed. The most recent format developed is one for machine-readable data files (MRDF). The analysis and review performed as the format was being compiled have resulted in questions being posed about the elements included in FIPS 30 (Software Summary for Describing Computer Programs and Automated Data Systems) and in a suggestion for a new standard to cover numeric data files.

Keywords: ANSI Z39.2; Bibliographic control; FIPS 30; Format structure; Machine-readable cataloging; Machine-readable data files; MARC; MRDF; Numeric data files; Software summary.

1. INTRODUCTION

In recent years, the importance of machine-readable data files or MRDF as an information resource has been recognized, largely because of economic factors. Costly duplicative efforts, ranging from writing new software for an accounting application to collecting and creating machine-readable statistical data on persons with Spanish surnames, can be avoided if one knew, in the first case, that numerous software packages are available commercially and are listed in trade product directories; in the second case, that such a statistical data file is compiled and distributed by the U.S. Bureau of the Census and is listed in the bureau's Directory of Data Files.

Bibliographic control in a library environment involves the "functions necessary to generate and organize records of library materials for effective retrieval." (1) By removing the word "library," this definition becomes more generalized, and it is within this broader framework that one can categorize the tasks involved in compiling the inventories or directories mentioned above as bibliographic control.

The purpose of this paper is to provide an overview of the standards associated with automated bibliographic control and the application of these standards at the Library of Congress to aid other institutions in their efforts to bring MRDF under bibliographic control as well as to relate this experience to a review of a Federal Information Processing Standards (FIPS) publication. Documentation is not discussed separately here because it is assumed to be an integral part of creating a machine-readable data file.

2. SUMMARY RECOMMENDATIONS

In connection with FIPS 30 (Software Summary for Describing Computer Programs and Automated Data Systems), questions were posed as to how certain elements were used and suggestions made for addition of other elements:

- Would it be useful to know the name of the person who wrote the software? Or the name of the organization, agency, or unit?

- Would it be useful to know for what agency the software was written, as in the case of contractual work?

- Are the software types, i.e., automated data system, computer program, subroutine/module, sufficient to describe the program being submitted? Or sought?

- Are the technical details, i.e., computer manufacturer and model, computer operating system, programming language(s), number of source program statements, computer memory requirements, tape drives, disk/drum units, and terminals, sufficient when looking for programs to acquire? Do users look for programs by these categories, e.g., programs written for an IBM 370 in OS?

- Would a thesaurus be useful in supplying keywords? In looking for programs of a certain type?

- Are there other kinds of data that should be provided to facilitate exchange or sharing of software?

Other activities in the federal government are leading toward standardization of elements to describe numeric data files. Development of a new FIPS for description of numeric data elements, taking advantage of the experience from the effort coordinated by the Office of Management and Budget, is needed.

3. STANDARDS FOR AUTOMATED BIBLIOGRAPHIC CONTROL

This section does not attempt to describe all existing standards for automated bibliographic control but instead discusses the ones having a significant effect on information processing. Actually, the impact of these standards has been far-reaching not only in this country but also abroad and in the public/private and for-profit and not-for-profit sectors.

3.1 Format Structure for Machine-Readable Bibliographic Data

The development of the American National Standard for Bibliographic Information Interchange on Magnetic Tape (ANSI Z39.2) took place in the late 1960s. The standard consisted of a format structure that facilitated the implementation of certain capabilities, such as providing a vehicle for the exchange of records so that only one set of conversion programs would be necessary, transcribing variable length data fields and variable length data records, being independent of any one manufacturer's hardware, and being independent of any single programming language. When the standard was first issued in 1971, these capabilities were considered innovative. Even with the considerable changes that have taken place in the computer industry and the data processing field from that time to 1979 when the standard was revised and reissued, these capabilities can still be viewed as moderately difficult to provide.

The structure consists of three components: the leader, the directory entries, and the variable fields. These are summarized below to give the reader an idea of the processing that would be involved when handling records using this structure (2):

Leader: A fixed field, twenty-four characters in length, occurring at the beginning of each record and providing parametric information for the processing of the record. Some of the parameters include the following:

Record Length: A five-digit decimal number equal to the length of the record, including the record length and the record terminator.

Status: A parameter indicating the relation of the record to a file, e.g., new, revised, or deleted.

Type of Record and Bibliographic Level: Two parameters used together to specify the characteristics and to define the components of the record. For example, Type of Record = Printed Language Material; Bibliographic Level = Serial.

Indicator Count: A parameter whose value in the form of a decimal digit is the number of indicators associated with each variable data field.

Identifier Length: A parameter whose value in the form of a decimal digit is the length of the delimiter plus the data-element identifier(s) used within the record.

Base Address of Data: A parameter whose value in the form of a five-digit decimal number specifies the character position of the character following the field terminator of the directory, where the origin is the first character of the leader.

Entry Map: A set of parameters specifying the structure of the entries in the directory as follows:

Length of the Length of Field Portion of Each Directory Entry

Length of the Starting Character Position Portion of Each Entry

Length of the Implementation-Defined Portion of Each Entry

(The tag is assumed to be three digits in length.)

The other positions in the leader are either reserved for future use or are reserved as implementation-defined positions.

Directory: A series of fixed fields or entries providing an index to the location of the variable fields within the record. The entries consist of the following:

Tag: Three alphabetic or numeric basic characters. Tags whose first two characters are zero specify the control fields; other tags specify data fields.

Length of Field: The length in the form of a decimal number of the variable field to which the entry corresponds, including the field terminator and any indicators. See also Length of the Length of Field Portion in the Entry Map above.

Starting Character Position: The character position, in the form of a decimal number, relative to the base address of data of the first character in the variable field referenced by the entry. The first character of the first field following the directory is numbered 0. See also Length of Starting Character Position in Entry Map above.

Implementation-Defined Portion: If present, contains information relative to the variable field referenced by the entry. See also Length of Implementation-Defined Portion in Entry Map above.

Variable Fields: Fields whose length is determined for each occurrence by the length of data comprising that occurrence. There are two kinds of variable fields:

Control Fields: Variable fields containing parametric or other data which may be required for the processing of the record and being specified by tags beginning with two zeros. Control fields are comprised of data and a field terminator; they do not contain indicators, delimiters, or data-element identifiers.

Variable Data Fields: Containing bibliographic or other data not intended to supply parameters for the processing of the record. May include indicators, delimiters, and data-element identifiers, as well as data and a field terminator.

Indicators: A one-character data element which is associated with a data field and which supplies additional information about the field. When indicators are

present, they are the first data elements in the field. See also Indicator Count in the Leader above.

Delimiter: Symbol that is used as an initiator, separator, or terminator of individual elements in a variable data field.

Data-Element Identifiers: When used, each identifier should be preceded by a delimiter, and each identifier should precede the data element it identifies. See also Identifier Length in the Leader above. The delimiter and data element identifier are combined to form a symbol used to initiate and identify data elements in a variable data field.

Field Terminator: A character used to terminate each variable field in the record. For the last data field, the field terminator is followed by a record terminator.

The structure embodied by Z39.2 has been adopted for different bibliographic applications, including those of the library community in this country and abroad and the scientific and technical community under the auspices of the Committee on Scientific and Technical Information (COSATI) whose format specifications are used by the National Technical Information Service, the Defense Documentation Center, the National Aeronautics and Space Administration, and the Dept. of Energy in their distribution of machine-readable records describing (mostly) technical reports and citations to journal articles. The library implementation of Z39.2, exemplified in the formats developed by the Library of Congress, is described in the following section.

At this point, it is important to note the reason for different bibliographic applications and hence, different implementations of the standard format structure. The library community in this country and in several other countries such as the United Kingdom, Canada, and Australia follows a set of rules and guidelines known as the Anglo-American Cataloguing Rules. NTIS, DDC, NASA, and DOE follow the COSATI Rules for Descriptive Cataloging. The tags and the contents of the fields, therefore, differ substantially as shown below:

COSATI Format		Library Format	
220	Unclassified Title	245$a	Short Title
240	Classified Title	245$b	Subtitle
260	Subtitle	260$a	Place of Publication
		260$b	Publisher
		260$c	Date of Publication

Even with these differences, the task of processing bibliographic records is simplified by the use of the standard format structure. When one considers the number of different bibliographic files that have to be processed by the commercial data base vendors (i.e., Lockheed, System Development Corporation, and Bibliographic Retrieval Services), the availability and application of such a standard have had tremendous implications.

Subsequent to its adoption by the American National Standards Institute, the format structure was submitted and adopted as a standard by the International Organization for Standardization as ISO 2709. A footnote here in terms of the relationship between ANSI Z39.2 and the Federal Information Processing Standards (FIPS): In the early 1970s, work had been started to incorporate the standard format structure into the FIPS series. There were, however, many objections raised by certain agencies because of a misunderstanding as to the relationship of the format structure to internal formats (the structure was for communications purposes only), the use (optional), and the scope of the records created using the format structure (dependent on the particular application, i.e., libraries were using it for books, serials, films, etc., and the users of the COSATI format were including technical reports and journal articles). In any event, the agencies that would have adopted the FIPS guidelines had already adopted and implemented the ANSI Z39.2 standard so the matter was dropped.

3.2 Other Standards

The distribution of machine-readable bibliographic records in the standard format structure was begun by the Library of Congress in 1969 and was probably one of the first major uses of the American National Standard Code for Information Interchange (ASCII). To provide the capability of representing all roman-alphabet languages and the romanized forms of nonroman-alphabet languages, the basic ASCII set of 128 characters was expanded to 180 characters by expanding the ASCII 7-bit code to an 8-bit code for 9-track tape users and contracting it to a 6-bit code for 7-track tape users. (The latter option was discontinued recently.) It should be noted that this developmental work had been taking place at a time when obtaining printouts using both uppercase and lowercase alphabetic characters was not easy. Shortly thereafter, IBM began offering a print train with the library character set as an off-the-shelf item. In recent years, several different CRT terminals have been developed with the capability of displaying the library character set in addition to COM (computer-output-microform) and photocomposition devices that can produce "printed" output with the full character set. Progress, however, has been much slower for languages in nonroman alphabets, both in terms of standard character sets and display/printing devices.

Other standards that were important in the development of the library applications included codes for country, states, and languages and representation of dates. Where a standard existed, it was adopted. In other instances, the library community worked with other agencies, which included the National Bureau of Standards because of its FIPS series, to develop the standard.

4. LIBRARY OF CONGRESS IMPLEMENTATION

In the preceding section, it was mentioned that the library community had developed an application for the standard format structure. This work had been initiated by the Library of Congress and has been referred to as MARC for machine-readable cataloging. In actuality, the Library had begun this effort as early as the mid-1960s and played an instrumental part in developing the format structure as a standard. The fact that the LC work on a series of formats preceded the development of the standard has caused some confusion in that the MARC formats developed by the Library of Congress are thought to be synonomous with the format structure ANSI Z39.2. This section describes the development of the MARC formats at the Library as well as the circumstances and the requirements being addressed in these efforts and the effects of these considerations on the format for machine-readable data files.

4.1 Development of LC MARC Formats

The MARC formats, as an implementation of ANSI Z39.2, were developed by the Library of Congress as part of a documentation package for a subscription service to distribute cataloging information in machine-readable form. This distribution process was a continuation of a program begun in 1901 for the Library to sell 3x5 cards containing cataloging information to the nation's libraries and was considered a major milestone in the progress of library automation. The formats and the dates of their publication or implementation include the following: books (1969), maps (1970), serials (1970), films (1971), manuscripts (1973), music (1976), and authorities (1976). Compilation of a format for machine-readable data files is in process.

From 1969 when the distribution service for cataloging information in machine-readable form was begun to the present time, the Library of Congress has created and disseminated over 2 million MARC records. When the service was first started, most of the subscribers could be categorized as individual library systems. Since then, the total number of individual subscribers has not increased dramatically, but the total number of users affected by the MARC service has been affected because of the following developments: the establishment of bibliographic utilities (organizations operating online computer facilities and maintaining large bibliographic data files to assist libraries in their bibliographic control activities); new services taking advantage of the availability of MARC records offered by commercial vendors, many of whom did not

exist before the advent of MARC; and bibliographic agencies in other countries offering various services to libraries in their own countries using the MARC records.

These developments have made the format maintenance process considerably more complicated. The formats have to reflect the changing requirements in the area of bibliographic control, but any changes made now have to be weighed in terms of their cost (particularly for the subscribers to the distribution service) as opposed to their perceived benefits. In addition, these developments have caused the MARC formats to become truly "interchange" in nature because institutions other than the Library of Congress are able to create their own machine-readable records and are in a position to transmit or distribute them to others. To perform these tasks, these other institutions depend heavily on the format specifications and other guidelines provided by the Library of Congress, and these agencies, therefore, have a vested interest in the formats produced by the Library.

4.2 Development of MARC MRDF Format

The MARC format for machine-readable data files is unique among its companion formats in that the Library of Congress has taken the responsibility to develop the format but has no expectation of collecting MRDF and placing them under bibliographic control in the near future. In other words, cataloging records for MRDF would not be originating from the Library of Congress. This would place the burden of identifying new requirements or changes in the format on the outside users.

In compiling this format, the Library followed a procedure instituted with the compilation of the earlier formats to appoint a working group of experts to assist with the work. Their expertise was heavily weighted toward statistical data files, partly because it was in this area from which most of the interest in bibliographic control was originating. The review process, however, was extensive throughout the library and information communities and entailed sifting and merging comments that were often contradictory from librarians who knew nothing about machine-readable data files, librarians who knew the MARC formats but nothing about MRDF, MRDF experts and data archivists who knew nothing about the MARC formats, and several other combinations.

Because bibliographic control of MRDF is still in its infancy, the process of identifying elements to be included in the format was not an easy one. The products (e.g., directories, data inventories, etc.) issued by agencies like the Bureau of the Census, the Geological Survey, the National Technical Information Service, or the national laboratories under the Dept. of Energy were consulted. During the review process, questions were raised about many of the elements derived from the inventories and directories, and these will be described in greater detail in the following section. There was, however, one problem that should be mentioned here. Machine-readable data files have been viewed, at least in the library world, as a single form of material when in fact they consist of at least four types (numeric files, text files, computer programs, representational, including graphic) and even more when one considers subcategories or combinations of these types. Each type of file has unique requirements that are not shared with the others. For example, information about the kind of computer system or memory requirements is essential to have for software but is usually unimportant for statistical data files. We had considerable difficulty in communicating to the library and information communities the needs of all types of MRDF users within the framework of a single document that was coherent to everybody.

5. RELATED FIPS ACTIVITIES

The analysis performed as part of identifying elements needed for bibliographic control of machine-readable data files has raised certain questions concerning standards issued in the FIPS series. This section discusses these problems with regard to FIPS 30 (Software Summary for Describing Computer Programs and Automated Data Systems) and proposes a new standard for numeric data files.

5.1 FIPS 30

This standard was used during the compilation of the Library of Congress MARC format for machine-readable data files to assist in identifying elements needed to describe software. Questions were posed, from a library orientation, as to how certain elements were used; in addition, it was felt that other elements that are usually provided as part of the cataloging process might also be useful for this standard.

- Would it be useful to know the name of the person who wrote the software? Or the name of the organization, agency, or unit?
- Would it be useful to know for what agency the software was written, as in the case of contractual work?
- Are the software types, i.e., automated data system, computer program, subroutine/module, sufficient to describe the program being submitted? Or sought?
- Are the technical details, i.e., computer manufacturer and model, computer operating system, programming language(s), number of source program statements, computer memory requirements, tape drives, disk/drum units, and terminals, sufficient when looking for programs to acquire? Do users look for programs by these categories, e.g., programs written for an IBM 370 in OS?
- Would a thesaurus be useful in supplying keywords? In looking for programs of a certain type?
- Are there other kinds of data that should be provided to facilitate exchange or sharing of software?

5.2 Need for New FIPS

Under the aegis of the Statistical Policy Division, Office of Information and Regulatory Affairs, Office of Management and Budget (formerly the Office of Federal Statistical Policy and Standards, Dept. of Commerce), activities have been proceeding to identify public-use data files available from federal agencies. These activities have culminated in the issuance by the National Technical Information Service of a Directory of Federal Statistical Data Files. The efforts involved in compiling this directory are described in a recent article by J. Timothy Sprehe of the Statistical Policy Division. (3)

One of the results of this activity has been a proposed directive from OMB for use by federal agencies involved in creating and distributing statistical data files. With a tentative title "Directive for Standardized Abstracts of Public Use Statistical Machine Readable Data Files," it would appear to be a logical starting point for a FIPS publication. Of particular interest is the fact that the proposed directive (and the directory compiled from data supplied by this directive) emphasize the use of bibliographic standards to facilitate the merging of these data with other bibliographic resources.

6. CONCLUSIONS

Bibliographic control is not an end in itself and cannot operate in a vacuum. With budget cuts, the sharing of resources, in this case of software, will probably accelerate in the next few years. It will be important not only to provide the necessary data to describe the software but also to be aware of the tools available that list software that could be exchanged or acquired. Workshops like this one are essential to bring the two players together: the creators of the software and the documentation and the outside users of the software and documentation. The creators will continue to create software and documentation to meet their own needs but with the knowledge that if they follow standards, they will facilitate the sharing process. Users of the (acquired) software should also be able to communicate their needs and problems in connection with the

packages they receive and the usefulness of the data provided in announcing software availability.

7. REFERENCES

(1) Dataflow Systems. A Glossary for Library Networking. Washington, Library of Congress, 1978. 34 p.

(2) American National Standard for Bibliographic Information Interchange on Magnetic Tape. ANSI Z39.2-1979. Revision of ANSI Z39.2-1971. New York, American National Standards Institute, 1979. 12 p.

(3) Sprehe, J. Timothy. "Implementing a New Federal Data Access Policy." Statistical Reporter, 81-12, 475-79, September 1981.

The Computer Program Abstract as Software Documentation*

Margaret K. Butler

National Energy Software Center
Argonne National Laboratory

The computer program abstract, while not universally accepted as a form of software documentation, has long been recognized as a necessary reference document by the computer user community. Users have banded together in cooperative organizations with like computing environments, common disciplines, or a shared need for particular applications software, to promote the use of computer program abstracts as a means of publication, program exchange, and software development cost savings. Compilations of program abstracts have been adopted to describe the contents of computer program libraries and to market the software industry's products.

In 1981, publication of an American National Standard for Computer Program Abstracts was approved, lending an air of respectability to the abstract's claim to a place in the software documentation family. This paper first seeks to define a computer program abstract and to point out its distinguishing features. Then, users and proponents of computer program abstracts are identified, and prior standardization efforts are reviewed, followed by a discussion of the development of the American National Standard.

Keywords: Computer program abstracts; Software documentation; Standards; Information systems.

1. INTRODUCTION

In most dictionaries you will find among the various definitions attributed to the noun abstract, something similar to the following:

> "a summary of a statement, document, speech, etc; that which concentrates in itself the essential qualities of anything more extensive or more general, or of several things; essence"

It is in this sense that the term computer program, or software, abstract has been applied to the form of software documentation discussed in this paper. A computer program abstract is a computer program summary, software documentation which conveys to the reader the essential qualities of the computer program, such as the nature of the information-processing functions performed or the physical problem solved and the computing environment required.

To better understand the position the abstract fills in the software documentation line, a number of characteristics peculiar to the computer program abstract are listed below. The abstract —

- functions as an introductory or "awareness" document. It is usually the reader's initial contact with the software.
- is the documentation most frequently overlooked by the author or software development project.
- is often omitted from the documentation requirements specified by the sponsor, or funding organization.

*Work performed under the auspices of the U. S. Department of Energy.

- is the primary concern of computer user organizations and government programs dedicated to "software sharing" and "technology transfer." Compilations of abstracts serve as reference catalogs to computer program libraries and software collections.

- appears routinely in the guise of a product announcement or description from computer manufacturers and software houses. In addition, catalogs of abstracts and software directories are marketed as information service products and reference volumes.

- is, by definition, a concise document readily transformed into machine-readable documentation.

- functions as a reference document. Its main purpose is to provide its readers, a segment of the computer user population, with sufficient information about the computer program for them to judge the appropriateness of the software to their needs and available resources. When compilations of abstracts are stored as computer databases, retrieval is improved by indexing and use of automated search strategies.

2. USAGE

Use of the abstract as a form of software documentation has been promoted for the most part by three categories of computer user organizations and the software and service components of the computer industry. The first category of user organization is the computer manufacturer's users group, typified by IBM's SHARE and Digital Equipment Corporation's DECUS. Computer program abstracts first found favor with these organizations. In 1954, when IBM set up an IBM 650 Program Library at its World Headquarters in New York City, all IBM 650 installations were encouraged to submit locally-developed software accompanied by an abstract describing their contribution. A year later when SHARE, the first computer users group, was formed by a group of IBM 701 users, one of its first acts was to establish a formal program exchange for its members with abstracts describing each program. As additional computer users groups came into existence, the other manufacturers agreed to staff and maintain similar program exchange facilities and to publish abstracts describing the contributed software, realizing the potential benefits of such information to their marketing operations. The importance of user-group exchange efforts has declined over the years with the "unbundling" of hardware and software, the emergence of a viable computer software industry, and the shift from predominantly scientific and engineering applications in a free-exchange, research and development setting to mostly commercial applications in a proprietary business environment.

The second category of computer user organizations consists of government agencies and contractors. The Atomic Energy Commission (AEC) and its successors — the Energy Research and Development Administration, the Nuclear Regulatory Commission, and the Department of Energy (DOE) — are in this class, along with the National Aeronautics and Space Administration (NASA), the National Technical Information Service (NTIS), and the General Services Administration (GSA). The AEC in 1955, in what was probably the earliest government action to promote software sharing, published a compilation of abstracts of available digital computer codes for nuclear reactor problems [1]. The agency and its successors have been publishing abstracts of agency-developed software ever since in the interest of software sharing and the elimination of redundant development costs. In 1960, the Argonne Code Center, a software exchange and information center, was created to handle the publication of abstracts and the maintenance and dissemination of nuclear reactor codes. The Center has since been renamed the National Energy Software Center (NESC), handles DOE-developed software, and the abstracts, in machine-readable form, have been added to the DOE Technical Information Center's RECON system.

In 1966, NASA established COSMIC, a center for the dissemination of NASA-developed software, at the University of Georgia. Government use of computer program abstracts was dominated until the seventies by the activities of these two centers and the AEC's specialized Radiation Shielding Information Center at Oak Ridge National Laboratory. Then in 1974, the Department of Commerce's NTIS, which is charged with making the results of federally-sponsored research and development available to the public, began to include abstracts of federally-generated machine-readable data files and software along with the abstracts of agency reports it publishes. Since 1977, NTIS has also published for the GSA the Federal Software Exchange Program catalog, which contains abstracts of the

federally-produced common-use software available to federal agencies, and state and local governments under that program.

Professional societies, trade associations, and other special interest groups, such as the American Nuclear Society, the Society for Computer Applications in Engineering, Planning, and Architecture, and the Quantum Chemistry Program Exchange make up the third category of user organizations who have endorsed the use of the abstract. As early as 1956, the Nuclear Codes Group (NCG), an organization which later became the Mathematics and Computation Division of the American Nuclear Society, adopted a standard abstract format for the use of its members, and in 1959, the NCG published a collection of their abstracts in the Communications of the Association for Computing Machinery [2]. A supplementary listing appeared in the same periodical the following year [3].

As the computer software industry developed, product announcements, computer program abstract catalogs, and software directories became a popular means of marketing the software vendors' wares. They appear, too, as products in their own right — the offerings of publishing and information service firms, such as International Computer Programs, Datapro Research Corporation, and Auerbach Publishers.

3. STANDARDIZATION

The initial computer program abstract standard was included in a computer program documentation standard developed by a study group composed of members of the American Nuclear Society's Mathematics and Computation Division. This group, identified as the STICE Committee for STudy In Cooperative Efforts, started out in mid-1965 to write a comprehensive documentation standard for scientific and engineering applications software. Because of a general reticence to impose too detailed and costly a documentation requirement on the program author and the developing installation, a significant fraction of the resulting Society standard was devoted to the computer program abstract [4].

In May of 1968, having explored standardization of computer program libraries at the behest of the Council of the Association for Computing Machinery, the Joint Users Group (JUG) Program Library Committee submitted a proposed standard to the American National Standards Committee Z39 on Library Work, Documentation, and Related Publishing Practices. Their proposal, "Field Headings and Codes for Machine-Sensible Computer Program Descriptions," was based on conventions defined for the submittal of program descriptions to the JUG Program Library Catalog [5]. The proposal was referred to SPARC/DOCN, an ad hoc Committee on Documentation set up under the X3 Standards Planning and Requirements Committee (SPARC) to study the need for and the feasibility, cost, and benefits of developing a national standard on program abstracts. As part of their study SPARC/DOCN designed a questionnaire to solicit the opinions of selected members of the data processing community. On the basis of the survey responses and the committee's deliberations, a suggested scope and program of work for an X3 standards development project on computer program abstracts, along with recommendations that a new technical committee be formed to develop the standard and that the JUG proposal be viewed as only one of a number of models to be considered, were transmitted to SPARC in late 1971. However, it was not until early in 1974 that Project 211 on Computer Program Abstracts was approved by the American National Standards X3 Committee on Computers and Information Processing, and it was September 1975 by the time Technical Committee X3K7 was able to start work.

During the 1971 to 1975 period, while little progress was made in the X3 arena, an extensive revision of the American Nuclear Society's documentation standard prepared by the Society's ANS-10 subcommittee was adopted as an American National Standard [6], and the leadership role in the federal government's ADP standards program was assumed by NBS and its Institute for Computer Sciences and Technology [7]. Federal Information Processing Standards (FIPS) Task Groups were organized to assist the NBS in the development, adoption, and implementation of needed standards. One of these, FIPS Task Group 14, assigned the task of developing standards and guidelines for the documentation of individual computer programs, automated data systems, and information processing systems, was given as its first priority the documentation of abstracts. The Task Group developed a standard software summary form, together with instructions, for describing computer programs and automated data systems for identification, reference, and dissemination purposes. The form and instructions appeared as a FIPS publication, FIPS Pub 30, dated June 30, 1974 [8].

The Committee spent many meetings refining the list of information items to be included in the standard and defining their content. A division of the list into "required" and "optional" items was investigated, as was the concept of a mini-abstract, or subset, to facilitate machine storage, search, and retrieval. Upon examination, because of the diverse needs specific to various segments of the computing community, both the mini-abstract concept and the rigorous classification of "required" versus "optional" items were abandoned. Instead, committee members agreed to the use of standard headings and a common set of items. Broad definitions were drafted for many items in an effort to encompass most of the frequently-expressed needs. The more parochial needs remaining were accommodated by allowing supplementary information items.

The sequencing of information items was discussed both in relation to the order in which items are defined in the standard and to the positioning of the supplementary items. A standard ordering is desirable from the user point of view, particularly when abstracts from a number of sources are being consulted. Several ordering methods were considered, such as library search techniques and classification schemes. The positioning of supplementary items in relation to the standard items was debated. It was recognized that while interspersing supplementary items would permit grouping of related information, all standard items must precede supplementary items for uniformity. Omissions of information are then readily apparent, and supplementary information is easily recognized.

In September of 1977 the sixth draft of the standard, containing the 24 information items shown in Table II, was circulated to over 500 identified users of computer program abstracts for review. Ninety-six of the reviewers responded with suggestions, praise, and criticism. All of the comments received were considered during subsequent X3K7 meetings.

Table II

DRAFT 6 INFORMATION ITEMS

1. Category	9. References	17. Operating System
2. Keywords	10. Programming Language	18. Related and Associated Software
3. Identification	11. Program Type	19. Communications
4. Abstract Date	12. Program Size	20. Material Available
5. Program Status	13. Processing Modes	21. Acquisition
6. Program Date	14. Timing	22. Distribution Restrictions
7. Narrative Description	15. Hardware	23. Contact
8. Method of Solution	16. Storage Requirements	24. Responsible Organization

A number of changes in the standard resulted from these discussions. The final document approved by the American National Standards Institute (ANSI) on March 9, 1981 contains just 22 information items; items 19 and 22 of the sixth draft were dropped and the definitions of other items expanded to include that information. The titles "References" and "Material Available" were modified to "Bibliographic References" and "Program Material and Support Available," respectively, to more specifically identify their intended contents. The order in which the items appear was revised slightly; "Bibliographic References" was placed in the position formerly filled by "Communications." Then, in 1978, the Committee decided to add an Appendix containing an abstract for a scientific application, and one for a business application, to illustrate use of the standard. Later, a third abstract for a computer manufacturer's software product was included to represent the software industry environment.

Copies of the computer program abstract standard have been available from ANSI since August of 1981. Upon adoption of the standard by the producers of computer program abstracts, the abstract should win acceptance as a form of software documentation, and when accepted, hopefully, two of the abstract characteristics presented in the Introduction can be changed to show the abstract —

- is the documentation most frequently prepared by the author or software development project.
- is always included in the documentation requirements specified by the sponsor, or funding organization.

At its first meeting, X3K7 reviewed the scope and program of work assigned the project and discussed the existing standards. Endorsement of the FIPS summary guideline, or a modification thereof, as an X3 standard was considered. To ensure that the standard proposed would accommodate the needs of the entire computing community insofar as practicable, the committee decided to collect and study the computer program abstracts in current use. A letter was sent to over fifty organizations and firms, known to produce abstracts or software summaries, requesting samples of their abstracts together with author's instructions or related material pertinent to the committee's needs. Sample abstracts were obtained from 46 producers; 29 of these were government sources. Seven examples were supplied by professional societies or other special interest organizations; five came from commercial firms and five from computer user groups.

Data taken from these sample abstracts were analyzed to measure the importance attributed to various information items by the abstract producers, to identify the headings, or titles, most frequently associated with the different information items, and to ascertain the applicability of the items to the four environments represented in the study. To do this, the titles, or headings of all of the information items used in the 46 sample abstracts were transcribed into machine-readable form and aggregated into 53 canonical headings, so that the number of occurrences of each heading could be tabulated as a measure of the importance of the associated composite information item. This did not encompass, however, those instances in which the information item was included as secondary information (i.e., subsumed under another title and different canonical heading). Those occurrences were tabulated separately. Table I shows the number of both primary and secondary occurrences for the abstract information items studied by X3K7.

Table I

IMPORTANCE OF ABSTRACT INFORMATION ITEMS MEASURED BY FREQUENCY OF OCCURRENCE

Title	Primary	Secondary
Available Material	14	
Category	23	4
Cost	11	2
Data Base or Data File	4	1
Date:		
abstract	10	1
program or revision	10	2
other	8	1
Documentation	17	3
Hardware:		
general	13	1
computer	28	4
disk/drum	2	3
memory	13	3
peripherals	9	1
printer	1	2
punch	1	1
reader	1	1
special	4	3
tape units	3	2
terminal	1	
Identification:		
general	3	1
abbreviated name	18	6
name	37	4
number	23	4
Input and Output	13	5
Keywords	6	3
Language	36	6
Method:		
general	11	3
accuracy	2	4
error information	5	1
math. functions	7	3
restrictions	17	4
Miscellaneous	8	
Narrative	46	
Organization:		
general	31	
author	30	
contact	15	4
installation	10	
user	7	
References	12	3
Security Classification	9	1
Size	22	
Software:		
general	18	1
associated	19	4
operating system	14	5
processing mode	2	1
unusual features	7	2
usage	9	
type	6	1
Space for Organization Use	7	1
Status	14	
Testing	2	4
Timing	13	1
Version	10	8

It is suggested that the ANSI X3.88 Computer Program Abstract be considered as a replacement for the present FIPS Pub 30. The format of the ANSI standard is more flexible. Additional information items identified as important to the abstract-user community are included, such as "Method of Solution," "Timing," "Related and Associated Software," and "Bibliographic References," and documentation of more detail is encouraged in other items, particularly in regard to the specific material available and the means of acquiring that material.

4. ACKNOWLEDGMENTS

Project 211 covering the development of American National Standard X3.88-1981 was carried out by ANSI Technical Committee X3K7, which I chaired over the five-year development period. Committee members who were major contributors to the effort and primarily responsible for the activities described in this paper are LaVon Boisen, Elizabeth George, Michael Landes, Helen McEwan, Betty van Gelderen, and Keith Wheeland. Grace Krause provided the secretarial assistance, technical typing skills, and diligence required to prepare the paper for publication.

5. REFERENCES

1. Radkowsky, A., Brodsky, R. A., Bibliography of Available Digital Computer Codes for Nuclear Reactor Problems, AECU-3078, October 1955.

2. Nather, V., Sangren, W., Reactor Code Abstracts, Communications of the Association for Computing Machinery, 2, 1, 6-32, January 1959.

3. Nather, V., Sangren, W., Reactor Code Abstracts, Communications of the Association for Computing Machinery, 3, 1, 6-19, January 1960.

4. American Nuclear Society Standard, ANS-STD.2-1967, A Code of Good Practices for the Documentation of Digital Computer Programs.

5. B. R. Faden, Computer Program Directory, New York, N.Y., Joint User Group (JUG) of the Association for Computing Machinery, 1971.

6. American National Standard, ANSI N413-1974, Guidelines for the Documentation of Digital Computer Programs.

7. National Bureau of Standards, Federal Information Processing Standards Index, FIPS Pub 12-2, December 1, 1974.

8. National Bureau of Standards, Software Summary for Describing Computing Programs and Automated Data Systems, FIPS Pub 30, June 30, 1974.

An integrated Machine-Readable Data Documentation System

Richard C. Roistacher *
Bureau of Social Science Research
Washington, DC 20036

The text formatter can serve as a powerful tool for manipulating text of all types. Techniques are described for using a single body of machine-readable text to generate questionnaires, data dictionaries, data editing materials, and archival documentation for machine-readable data files.
Keywords: Data documentation; text formatters.

1. TEXT FORMATTING PROGRAMS

An indirect text formatting program with a macrocommand facility and the ability to link to high-level language subroutines can be used as the basis of a generalized documentation system. Although the University of British Columbia's Format program is used in the implementation described here, any other formatter with the necessary characteristics will do equally well.

1.1. Indirect text formatters

An "indirect" text processor is one which accepts as input a file containing text and formatting commands, and produces an output file of formatted text. An indirect text formatter can be contrasted to a "direct" or "what you see is what you get" formatter, which produces formatted text as it is entered. (Direct formatters are seen only on micro computers or dedicated word processors.) An indirect text formatter has both global and local commands. An example of a global command in Format is "width 65", which sets a text line width of 65 characters from the point of invocation to the end of the document or to the next "width" command. An example of a local command in Format is "/1/", which indicates that a new line is to be started at that point. Unlike many indirect text processors, Format does not require that local commands be in any particular position in the input line. Many text formatters require that all command operands be on a line by themselves.

1.2. Macros

Format allows the definition of macro commands with substitution of up to ten parameters. An example of a macro command with an argument is "|h2" which generated the heading for the first section of this paper. The macro is defined

```
/w612n/ |count(h2)|.  /C/|par.1| /nC/ |contents(|par.1|,1)
```

An example of the macro's invocation is

```
|h2('Text formatting programs')
```

The expansion of the macro puts the definition into the program's input stream with the text argument substituted for the parameter string in the definition. The macro starts a new page if less than six lines remain on the current page ("w6"). If not at the top of a page, it skips two lines ("12"), begins centering ("n"), prints and then increments a counter named for the heading ("|count(h2)|."). The macro then prints the argument converted to upper case ("C"), turns off uppercase conversion and centering and starts a paragraph ("Cmp"). Finally, the macro makes an entry into the table of contents.

*The author was unable to present this paper at the workshop.

1.3. Functions

Although the syntax of a Format function call is similar to a macro invocation, a function is quite different in definition and operation. Most format functions are compiled FORTRAN subroutines which are dynamically loaded at execution time. The contents function builds a queue, each of whose elements is a macro containing the text, the current page number, and a number indicating the level of indentation of the contents item. At the end of the document, the contents queue is fed into the input stream and expanded to yield a table of contents. The ability to load FORTRAN subroutines allows for an almost arbitrarily great expansion of the text formatter while maintaining its original syntax. Built-in format functions include such things as calls to the system time and date routines, a variety of alphabetic and numeric counters, and stack utilities such as tables of contents and indices.

1.4. Libraries

Document processors usually have both built-in and user-supplied libraries of macros and functions. The ability to handle multiple libraries is helpful, in order to define alternate expansions of macrocommands.

2. MACRO AND FUNCTION USAGE

The original use of macros was as a shorthand, to avoid the repetition of long strings of commands in recurring items such as headings. It is certainly easier to type "|h4" than the long string of commands and function calls. In addition, a set of macros imposes a stylistic discipline. A well chosen set of macro names allows an easy transition between related syntactic forms. Thus, although a level three heading may be quite different in syntax from a level four heading, the manuscript change from one to the other is simple and straightforward.

Finally, and most important, the invocation of the macro can be separated from its definition. The American Psychological Association uses four levels of heading which differ in capitalization, centering, etc. Other learned societies employ different conventions, e.g., the numbered paragraphs used in this paper. In most cases, the logical structure of a document is independent of the stylistic conventions used in its publication. Thus, the arbitrary macro names "|h1", "|h2", "|h3", and "|h4" can be expanded according to any manual of style. In cases where a stylistic format does not allow four levels of expansion, two levels which are to be merged may simply share the same definition. The object is to eliminate the need for editing the original document. The output format is determined by the library of macro and function definitions supplied to the text formatter at the time the document is processed.

3. FILE DOCUMENTATION IN SURVEY RESEARCH

A significant portion of the Bureau of Social Science Research's work is in survey research. The basis of a data file is usually a questionnaire. The questionnaire is developed by the principal investigator, who usually seeks help from the field staff and the coding staff in developing question formats, coding schemes for responses, and the skip patterns. Prior to the development of the system described here, the questionnaire was typed by hand or produced on a word processing system. Most questionnaires are designed for keying from the original instrument, rather than for transcription to coding sheets. In some cases, open-ended responses must be converted to numeric code before keying, but in most cases record position numbers and missing data codes (for blank responses) are printed in the right margin of the questionnaire. In most cases, the questionnaire went through several drafts before being submitted to computing services for the assignment of record position and missing data information.

3.1. Data dictionaries

BSSR processes most of its survey data files with OSIRIS IV, a hierarchical data management and statistical system developed at the University of Michigan. OSIRIS IV, like all modern data management systems, requires that data be defined by a machine-readable dictionary. Usually, the writing of a dictionary description was deferred until the questionnaire had been sent to the printer. A computing specialist used a paper copy of the questionnaire as a guide for keying a file giving each data items identifier, name, location, width, missing data indicators, etc.

3.2. Editor's codebook

Many questionnaires contain open-ended items which are coded after the interview, but prior to keying. A question such as "How do you spend your spare time?" may have more than one answer and may include responses which are entirely unforeseen at the time the questionnaire was designed. In such cases, a set of blank lines is printed on the questionnaire and eight to ten columns allocated in the record. The expectation is that there will be no more than 99 categories of answer and that no one will mention more than four or five activities. The actual allocation of the columns into variables is done after the coding scheme has been established. The occasional occurrence of responses requiring more than the allocated space results in some extra work to add one or more data items to the end of the file. Usually, the editing codebook was constructed by cutting and pasting a questionnaire into a question-per-page format.

3.3. Analysis and codebook

Once the file has been cleaned and scales and indices constructed, it is passed to the analyst. The analyst usually uses an abbreviated codebook containing only the variable identifiers, labels, and category values and labels. The statistical system produces a formatted version of the machine-readable dictionary which is most often used as the analyst's codebook.

3.4. Archival user's guide

An increasing number of statistical files are converted to an archival or public use form. Such archival files require a summary form of documentation which will relieve secondary users of the necessity of consulting a file's primary users. An archival user's guide usually contains an account of the project's adjectives and history, a history of data collection and file processing activities, and a codebook for the machine-readable data file.

The archival codebook contains the complete text of questions and values, labels, and explanatory text for categories. When possible, an archival codebook also contains frequency accounts for each category of a discrete variable and means, variances and ranges for continuous variables. A variety of systems has been used for the production of archival user's guides. In some cases, the machine-readable dictionary can be augmented with additional text to produce such a document. However, in most cases, user's guides are produced *de novo* with typewriters or word processing systems.

4. THE BSSR INTEGRATED DOCUMENTATION SYSTEM

The integrated documentation system began as a set of macro definitions and functions for the formatting of questionnaires. A limited set of syntactic forms was devised and a set of Format macros was developed to produce those forms. Formats counting functions, as well as some especially written functions were used so that all seriation is done by the document processor. Thus, question numbers, category values, and record locations are all supplied by the document processor. Questions may be inserted, deleted, or moved about at will while maintaining complete integrity of seriation.

Figure 1 shows a fragment of a questionnaire manuscript; Figure 2 shows the fragment as formatted by the document processor. All seriation has been produced by Format. The thirteenth use of the "|q" macro has produced question 13 in the questionnaire. The categories have been numbered serially from 1. The "|col" function indicates that one column is to be allocated for the result and that a missing value is to be entered as "9". Since question numbers are not known until execution time, questions referred to in the text are given labels, which the formatter resolves to question numbers. In this case, the question is labelled "A9" and has skips to questions "A14", "A37", and "C31". In the printed questionnaire, all labels have been resolved to question numbers.

```
|q(A9) On most political matters, do you consider yourself:
|cskip('Liberal',A14)
|cskip('Moderate',A37)
|ccont('Conservative',C31) |col(1,9) '
```

Figure 1: A questionnaire manuscript fragment.

```
13.  On most political matters, do you consider
     yourself:

      Liberal (SKIP TO Q. 21) .....................1

      Moderate (SKIP TO Q. 28) ....................2

      Conservative (CONTINUE WITH Q. 14) .........3   18/9
```

Figure 2: Symbolic labels resolved and manuscript formatted.

4.1. Data dictionary description

The questionnaire formatter contains much of the information needed for defining an entry in the data dictionary, e.g., location, width, and missing data code. The questionnaire manuscript was augmented with a number of functions needed for the production of dictionary information. Figure 3 shows the fragment from figure 2 augmented with a short variable name, a second missing data code, a file identifier, and a reference number.

```
|dict('Liberal or conservative') |dmd2(4) |did(CBS) |drefno(1014)
|q(A9) On most political matters, do you consider yourself:
|c('Liberal',,A14)
|c('Moderate',,A37)
|c('Conservative',,,C31) |col(1,9)
```

Figure 3: Questionnaire manuscript augmented with dictionary information.

Figure 4 shows a line of the set-up for the OSIRIS data definition program. This setup is used to define the field in the data record which will hold the information from question 13. The output from the dictionary definition program is a machine-readable data dictionary for the questionnaire data file. The dictionary allows OSIRIS IV and other statistical systems to read the data file, take appropriate action to cope with missing data, and label the output of analysis programs.

```
VAR=14 NAME='Liberal or conservative' COL=18 WIDTH=1 REFNO=1014 -
MD1=9 MD2=4 ID='CBS' L='1=Liberal,2=Moderate,3=Conserva'
```

Figure 4: OSIRIS IV dictionary description record generated from the manuscript fragment.

The dictionary information not explicitly supplied in the manuscript defaults to commonly used values. The variable number is incremented by one for each succeeding line of variable identification. All defaults can, of course, be overridden by explicit entries in the manuscript. Figure 5 shows the same fragment formatted as part of an archival user's guide.[1] The statistical system was used to generate a file of frequencies which is merged with the codebook to produce the printed frequencies to the left of the category labels.

```
V14                              Reference:   1014
Liberal or conservative          File I.D.:    CBS
Location: 18 Width: 1            Numeric character
Missing Data: EQ 9 OR GE 4

On most political matters, do you consider yourself:

     405    1  Liberal
     437    2  Moderate
     331    3  Conservative
      42    4  Missing data
```

Figure 5: Manuscript expanded into user's guide.

4.2. Further extensions

We plan to produce a macro library which will format the manuscript for input to an information retrieval system. system. For this extension, the manuscript would be augmented with a "|keyword" function with retrieval keywords as arguments. The function would be ignored in the formatting of the questionnaire and would be printed in the usual keyword format for the user's guide. The keyword function would be appropriately redefined to provide proper input to the information retrieval system.

5. COST AND SCHEDULING BENEFITS

The original purpose of the documentation system was to save time and effort in the preparation of questionnaires. Satisfactory results were obtained in the first use of the system, when multiple versions of a 325-question instrument were prepared. The original version of the questionnaire required six weeks for preparation, (including time for debugging of the formatting system). Preparation time for successive versions went from one day to four hours.

An unforeseen effect of the system was to relieve the data processing section of some of its time pressure. Usually, slack in the schedule is consumed during the earlier phases of the project, with most of the extreme rush occurring during computing, analysis, and writing. As the system developed, it became clear that the use of an integrated documentation system was improving the work flow throughout the questionnaire and file construction process. Variable and category labels, as well as dictionary information, could be inserted at the time the questions were drafted, thus moving dictionary definition work several months forward. The use of an integrated system also made questionnaire authors aware of problems which might arise during file definition and construction. The system yields savings in time and labor, production of better questionnaires and files, and a beneficial redistribution of the work sequence.

6. IMPLICATIONS FOR STANDARDIZATION

The work described here has implications for three kinds of standardization. that of content, output format, and input format.

6.1. Standardization of content

Standardization of content is the most difficult of problems, because it must be applied anew to each new work. To require a "complete and concise abstract" is only to hope that the author will write such an abstract this time, just as last time. Some items of content are constrained by technical factors. Codebooks produced by the system described here may not include variable names, but all will have field locations and widths, for without such items, the system cannot produce even an empty codebook. It is possible to attempt the enforcement of content standards by technical means. A documentation system might forbid the use of strings of blanks as variable names, or might require that an abstract contain at least 240 non-blank characters. (The reader's speculations as to the efficacy of such restrictions are as least as good as mine.) My feeling is that standardization of content can be enforced within a contractual relation, but not necessarily within a community.

6.2. Standardization of output format

Since the output format is entirely imposed by the documentation system, the format of all automated documentation is, ipso facto, standardized. The problem of standardizing an output format thus reduces to one of convincing those with responsibility for documentation systems to support a standard format. Once written into a documentation system, the standard format is automatically imposed on the authors of documentation.

6.3. Standardization of input

The input and output of a documentation system are equivalent in content, but differ greatly in form. The question therefore arises as to whether a standard format should be imposed on the input to documentation systems. My feeling is that such standardization efforts would be unduly restrictive and would be ignored. Text processing languages are being developed at a great rate. People are developing new text processing procedures and languages, which they will promulgate with a fine disregard for standards and restrictions. The time for standardization of text processing languages has not yet arrived.

The lack of a standard for text processing languages is ameliorated by the relative ease with which a manuscript file in one language can be translated to another language. Since all formatting information is in a regular form, there is usually little difficulty in writing a procedure which will do much of the translation work. Translation beyond the power of a straight forward program usually involve differences in the capabilities of the source and target documentation systems. The resolution of such differences usually involves a considerable editorial touch.

7. REFERENCES

1. Roistacher, R. C. A style manual for machine readable data files and their documentation. Washington, DC: U. S. Government Printing Office, 1980.

Compilation of Bibliographic Data Element Dictionaries

Madeline M. Henderson

Consultant, Bethesda, Md.

Under sponsorship of the National Technical Information Service (NTIS), a project has been started to compile a data element dictionary (DED) encompassing all the bibliographic data elements used in the processing of Federal documents. These elements include those used in the abstracting-indexing type of processing done by the four major reports-processing agencies (NTIS, Department of Defense, Department of Energy, and National Aeronautics and Space Administration), plus those used in the library-type of cataloging done by the Government Printing Office, with the Library of Congress, based on the family of MARC formats. The purpose of such a DED is two-fold: first, to provide to the reports-processing agencies a tool to guide their consideration of possible further standardization to achieve greater compatibilities and improved cooperative processing among themselves; and secondly, to provide to them and their library counterparts a mechanism by which to explore productive avenues of cooperation and interfacing. This DED is one of several similar efforts now underway or recently completed; the formats used and the experiences gained should be usable by other efforts, perhaps through preparation of guidelines.

Keywords: Bibliographic data; Data element dictionary; Guidelines.

1. INTRODUCTION

The data element dictionary, as defined in this paper, is a compilation of descriptions as to the meaning, contents, and rules for use of the units (elements) specific to a particular data base. The dictionary catalogs and defines the data elements; it does not locate the data within the organization's computer system nor process information about data entities and associated data processing functions. In this paper, the data element dictionary is a more narrowly-defined management tool than the data dictionary system for which the Institute for Computer Sciences and Technology has proposed guidelines [1] and a standard [2]. However, the dictionary can serve as one step in the process of improving data management and therefore deserves consideration for preparation of user guidelines.

1.1 Purpose of Current Project

At the present time, four Federal agencies -- the Department of Defense's Technical Information Center (DTIC), the National Aeronautics and Space Administration's Scientific and Technical Information Facility (NASA/STIF), the Department of Energy's Technical Information Center (DOE/TIC) and the National Technical Information Service (NTIS) -- perform the major tasks in the bibliographic processing of the technical reports emanating from Federally-conducted or -sponsored scientific and technical research and development activities. The processing involves recording the pertinent data identifying the reports, abstracting and indexing the salient information contained in the reports, and recording the means by which the reports can be obtained from one or more of the above-mentioned agencies.

The four agencies have developed computer-based systems to process the reports pertinent to their own mission responsibilities, and to produce various forms of documentation services based on them: printed abstract bulletins and indexes, machine-readable data bases, specialized alerting services and bibliographies, etc. The computer services are unique to each agency in terms of specific hardware and software, but the exchange of bibliographic information among them is based on the ANSI Standard Z39.2-1971 (updated in 1979) format for bibliographic information interchange on magnetic tape.

That ANSI standard has been implemented in two major formats -- that employed by the reports-processing agencies identified above (the COSATI implementation) and that used by libraries in sharing cataloging data across a family of special formats (the MARC formats). These two systems have much in common but many fundamental differences -- the philosophy underlying each implementation is specific, imbedded in its own community of users, based on years of practice, and well-nigh immovable against change or casual modification.

Given this environment, a project to develop a data element dictionary encompassing both philosophies, i.e., both sets of data elements, seems to fly in the face of prudent activity. But the communities agree that now is a good time to examine and evaluate the interfaces between these two philosophies/systems, as well as the characteristics of the systems themselves.

Dating back to 1978, the possibility has been discussed of developing a data element dictionary (DED) of the data elements, with subsets of codes and rules, used by the reports-processing agencies, with the purpose of investigating needs for better standards and protocols so as to contribute to lowering the costs and increasing the speed and efficiency of Federal reports-processing operations. Present procedures of the four agencies need to be reviewed with a view to determining the feasibility of formulating a network concept and the impacts and benefits of such a concept on each agency and the user community.

More recently, the possibilities have been extended to include the data elements of the library community's cataloging operations. Recent developments in computer processing of bibliographic data have included cooperation in inputting that data and more extensive on-line access to the resulting files of data. These developments have been particularly successful in the library community. The analysis of elements in a DED for reports-processing should be undertaken, it is felt, in light of the successful activities underway or projected in the library community, in order to improve the interfaces and interactions between these two segments of the total information processing community.

The current project, then, seeks to compile a DED encompassing all the data elements used in the processing of Federal documents: the abstracting-indexing operations of the four agencies plus the library-type cataloging done by the Government Printing Office (GPO). The purpose of such a DED is not only to provide the four agencies a tool to guide their consideration of possible improved cooperative processing among themselves, but also to provide them and the library community a mechanism by which to explore productive avenues of cooperation and interfacing.

1.2 Relation to This Meeting

The Institute for Computer Sciences and Technology (ICST) has initiated discussions of a standard for data dictionary systems (DDS), defined as computer software "used to record, store, and process information about an organization's significant data and associated data processing functions." [1] A fully developed data dictionary for an organization is defined in ICST's prospectus for a standard [2] as a catalog of the organizations' data resources but also "a computer data base that fully documents its data collection, processing, handling, and dissemination activities." The DED efforts described here do not constitute as extensive a management tool as the DDS of concern to ICST; however, the DED can benefit from the same considerations as to data standardization and control.

The DED can help the agency (or agencies) whose data are described to "reduce redundant data collection and to improve the utility of existing data resources." [2] The DED will highlight similar data elements and suggest actions to improve effectiveness and reduce costs in redundant data gathering.

Standardization of the data elements unique to each agency and of those used across all the systems can improve the collection and sharing of data among agencies and systems, particularly between the two types of systems or philosophies as described earlier. A DED can assist efforts to develop acceptable standards as appropriate to the overall bibliographic processing effort.

The DED, as is true of a DDS, can assist in the definition of new systems or, even more importantly, during the upgrading of existing systems. The DED can be useful in determining the effect of proposed changes in the total operational environment.

From these factors, it is apparent that a DED as defined here is an important component of a DDS as defined in ICST documents. As such, the DED will benefit from efforts to define data element sources, descriptions, and pertinent rules for usage within the bibliographic data processing communities, leading to guidelines for a tool for improved management of computer files and data bases within and by that community.

2. CURRENT DEVELOPMENT EFFORTS

In the process of defining and establishing the project described above, to compile a dictionary of the bibliographic data elements used in the processing of Federal documents by two parallel but separate processing systems, a number of related efforts were examined. Their experiences and directions were valuable in guiding the design and initial activities in the current project.

2.1 DTIC Uniform Data System

Of particular interest was the data element dictionary compiled by DTIC in its development of a uniform data system. As noted in the preface to the published dictionary, DTIC currently "maintains four separate information banks that operate independently of each other yet contain parallel information." [3] The DTIC technical report-processing system as presently operational is one of the systems to be included in the NTIS project. DTIC's objective in compiling its DED is to provide capability of standardized access to all like data; for this purpose, standardization of data entry, application, and retrieval are required. Such standardization and the restructuring of the individual data bases to achieve a uniform system makes a specific data element required in one data base available for use in another, if appropriate.

The DTIC data element dictionary was seen as an authoritative document to define the elements and their specific uses, to serve as a communication link between system designers and users, and to provide an effective management tool for DTIC's information systems.

The stated purposes of this dictionary paralleled those of the NTIS-sponsored project, at least in part: to define the data elements used in several information banks which are independent of each other and yet basically similar in purpose and content, to serve as a communication link among those responsible for the individual systems and those who use them, and to provide a useful tool for the managers of the individual systems in reviewing their unique system characteristics and for their collective consideration as to possible standardization and community activities.

Of course, DTIC started out with the avowed intention of standardizing across the separate systems for purposes of cost effectiveness and efficiencies; the NTIS project offers only the means to consider possible standardization for purposes of further cooperative activities -- which can lead to improved effectiveness and efficiency as a long-range goal.

The format of the DTIC dictionary has proved useful to the current project, as has its list of descriptive units for each data element.

2.2 UNESCO Data Element Directory

Another pertinent effort is the UNESCO-sponsored compilation of lists of data elements in six formats used for the computer-based exchange of bibliographic data between/among national bibliographic agencies. [4] The purpose of the document is to provide a tool to facilitate analysis to determine which data elements in the various formats are sufficiently similar as to prove hospitable to standardization and which are not amenable to such standardization, either because recording practices differ too widely or because an element exists in one format and not in another.

One comment in the introduction to the document is particularly pertinent to all similar data element compilation efforts, including the current project: "An examination of this document will be sufficient to assure the reader that the analysis of existing data elements, and the arrival at a single set of elements for the Common Communication Format, will not be an easy task." However, the first steps must be taken and the analyses attempted, in order to accomplish whatever agreements might be possible.

Again, the format of this DED and the list of descriptors about each data element were particularly useful in the design of the current project. Experience in the UNESCO study has proved, for example, the value of an index to each system's list and an overall index to the compilation. Originally it had been thought that providing comparable information in an identical format would be sufficient to facilitate comparison and analysis. However, variations in terminology make it difficult to know where (or even whether) to find related data elements in the separate lists.

2.3 Archival and Manuscript Records Dictionary

A project currently underway, sponsored by the Society of American Archivists, involves the compilation of a dictionary of data elements for more than 20 archival and records holdings systems. The list of information about data elements collected includes descriptors about the data element itself, plus descriptors about the system in which the data elements operate: system limits as to number of characters per record, for example, number of fields per record, length of field, etc. These characteristics were needed, it is said, because the purpose of the data element dictionary is to provide means not only for standardizing the data elements themselves but also for exchanging data among the systems.

Most of the participating organizations chose to report about their data elements in machine-readable form, following a data collection form developed for the project. The data were "recorded in a simple word processing/file management system on an in-house mini for report generation purposes," according to the project manager.

This project represents a more extensive data gathering activity, across more files, than the others described above; however, the number of descriptors about the data elements is similar to that of the other efforts.

2.4 The NTIS-Sponsored Project

The purpose of the project, to compile a data element dictionary for the bibliographic data used in processing Federal documents, has already been described. The effort at this time involves the development of the data gathering form and the definition of machine requirements as to format and recording media. The computer system at GPO has been offered for the collection of descriptions about the data elements and for the generation of the total compilation and accompanying indexes, as well as specialized sub-reports such as numeric listings of tags used, for example. The computer system at DTIC has also been offered; at this time details on specific machine requirements are being compared and resolved.

The descriptors to be recorded about each data element in each system represent a combination of those already used in the various efforts described above. More descriptors about the elements themselves will be recorded than were used in the DTIC and UNESCO ventures, but fewer items about the computer systems will be captured than in the archives project.

The current project differs also from some of the other efforts in that the data elements will be identified, defined, and recorded by each of the participating agencies rather than by a centralized activity. This decision is based on the feeling that the need to so identify data elements will be a useful exercise for the system managers. The DED project at Defense has already been described; NASA/STIF has just initiated a review of its data entry procedures, so the DED effort will complement that program also.

3. POSSIBILITY FOR GUIDELINES

The NTIS-sponsored project to compile a data element dictionary of the bibliographic data elements in the processing of Federal documents represents one example of a number of similar efforts. These constitute a recognition by system managers of the value of compiling and reviewing a system's data elements in a structured manner. As noted by ICST, "agencies are striving to reduce redundant data collection," because of increasing costs of labor-intensive data handling. In addition, a DED can support the development of data standards, so that data collected for one purpose can serve another as needed by various activities.

Review of the current project and the related efforts described above can lead to guidelines about the bibliographic data elements which should be described, the extent of descriptions about each element which will yield valuable information for management decisions, and the type of machine processing of the descriptions which will produce the most effective tools to aid in making those decisions.

For example, the guidelines would define the bibliographic data elements which should be captured for any individual system for which a dictionary is to be compiled. Further, the guidelines would list the minimum number of descriptive items which should be recorded about each bibliographic data element in order to compile a dictionary of such elements: name of the element, narrative definition, rules for use of the element, source (i.e., system in which it is used), character set permissible, tags/indicators used with the element, etc. Some nine of these descriptive items have been defined for all four current efforts described above.

The guidelines would also describe the steps in machine processing to yield the most useful dictionary: designation of fields to be used in preparing indexes to the dictionary listing, which items are best input in fixed length fields and which must be allowed variable length input, and the format found most useful in both the dictionary entries and the indexes to the dictionary.

Development of guidelines is best done through appropriate standards-setting bodies such as ANSI Committee Z-39 on Library and Information Sciences and Related Publishing Practices. If such guidelines are completed, they could be offered simultaneously as FIPS Guidelines for Federal use.

It needs to be said again that the DED is a step in the process of improving data management; the DED can guide system managers about internal operations and cooperative activities with other systems. But the decisions as to data standards, for example, are a step beyond the compilation of a DED. What is suggested here is preparation of guidelines for the compilation step only.

Further standardization procedures will not be easy to accomplish. The existing systems represent a major investment of resources in hardware and software, in procedures and processes, in products and services that exist and fulfill the mission-required responsibilities of the agencies. To talk of changing those systems to meet community-imposed standards causes grave concern, and rightly so, on the part of the system managers. As noted in the ICST documents, the utility of a given capability for a sufficiently important segment of the community, in view of known constraints, must be determined before standardization efforts can be successfully undertaken.

This becomes especially important when the suggested standardization encompasses two distinct system types or philosophies about the bibliographic processing of Federal documents. There is a tendency to fear being subsumed or "swallowed up" by one approach or the other. This fear has been expressed, in particular, about the abstracting-indexing systems of the four reports-processing agencies because of the more pervasive cataloging system and numerous MARC-based formats used by the library community.

4. CONCLUSIONS

What is needed, then, to sustain the momentum toward improved efficiencies and effective coordination is to complete the compilation of the dictionary of bibliographic data elements used in processing Federal documents. That effort itself will help the individual system managers to recognize the current status of their data elements and may suggest changes or modifications for the individual systems.

The compilation, further, will point up similarities and differences in the handling of common data elements and may suggest areas for discussion and compromise leading toward improved efficiencies in the overall procedures for bibliographic processing. One projection calls for the five systems to try to reach agreement and standardization for handling a set of core data elements common to all systems, so that initial processing of Federal documents can serve the different needs of the individual systems for specific products and services. Thus sharing of bibliographic input can result in sharing of information resources, with accompanying savings and improved timeliness for users of the systems.

5. REFERENCES

[1] Federal Information Processing Standards Publication (FIPS PUB) 76. Guidelines for Planning and Using a Data Dictionary System. Washington, D.C., U.S. Dept. of Commerce, National Bureau of Standards, 1980 August 20, 10p.

[2] Institute for Computer Sciences and Technology, Prospectus for Data Dictionary System Standard, NBSIR 80-2115, Washington, D.C., U.S. Dept. of Commerce, National Bureau of Standards, September 1980, 19p.

[3] Kuhn, Allan and Melissa L. Young, Compilers, Data Element Dictionary; DTIC Uniform Data System, DTICH 4185.8, Alexandria, Va., Defense Logistics Agency, Defense Technical Information Center, April 1980, 551p.

[4] Simmons, Peter, Compiler, Data Element Directory, Paris, Unesco General Information Programme, 1979, 1400p.

Capital games: the problem of compatibility of bibliographic citations in data bases and in printed publications

Hans H. Wellisch

College of Library and Information Services
University of Maryland

Bibliographic citations taken from the major databases cannot be used in their original form for publication in most American journals because of editors' capitalization practices. These are based on rules invented a long time ago that run counter to normal English orthography and are based merely on tradition. They have no justification from a linguistic point of view, and may even make titles ambiguous. The relevant American Standard ANSI Z39.29, based on the International Standard ISO 4, allows two different styles for titles of monographs or serials, but only one--normal orthography--for titles of articles or papers. Most major databases also use normal English orthographic rules for titles, but some render titles in all capitals, a practice which may also be detrimental to rapid exchange of bibliographic data among different media. While printers may display titles any way they like, journal editors and database producers should adhere to the commonly accepted orthographic standard (followed also by the Library of Congress), namely capitalization of only the first word and all proper names or acronyms in a title. This would make it possible to transmit bibliographic data for references or footnotes directly from a database without time-consuming and error-prone re-editing for capitalization.

1. INTRODUCTION

The generation and display of bibliographic data in databases as well as in conventional printed publications suffers at present from a complete lack of uniformity which is due to three different conditions of standardization:

(a) the existence of a multiplicity of standards which are either overlapping, mutually contradictory, or applied in sometimes quite different ways by the producers of recorded information;
(b) a lack of standards, resulting in idiosyncratic rules and solutions of certain problems; and
(c) the persistence of obsolete quasi-standards, invented by unknown people on their own authority, sometimes in a distant past, when they were geared to publication methods and patterns that have now themselves become entirely obsolete.

Condition (a) has been very well analyzed and summarized in two papers [1, 2] which unfortunately appeared in rather out-of-the-way or limited-distribution publications, but deserve close study by anybody interested in international standardization of bibliographic data. Condition (b) will be discussed at and perhaps ameliorated by the results of this workshop. Here, I wish to address only one seemingly minor or even trivial aspect of condition (c), namely the capitalization of words in titles of monographs, serials, and individual articles and papers (the latter forming the vast majority of all bibliographic references in data bases). I was recently made aware of the adverse effects of quasi-standards pertaining to capitalization when I had to write a review article, referring to a large number of books and articles. Many of the references came from searches performed on several databases, both printed and electronically accessible. Most titles in the printed sources as

well as in computer printouts were displayed in normal English orthography, i.e. only the first word and proper names or acronyms were capitalized. References found in library catalogs, all written in the style of the Library of Congress, also followed the same pattern. But when the manuscript was submitted to the editor, every single reference had to be reworked manually so as to conform to arcane rules of capitalization followed by the journal. This was a boring, tedious, time-consuming, and entirely meaningless and illogical task. I asked myself: is this not a terrible waste of time and energy, expended only to satisfy the whims of people now long since dead and forgotten who dreamed up rules of capitalization in the horse-and-buggy age? Is it really necessary or useful to do this now that bibliographic data can be generated at one focal point, transmitted electronically to one or more databases, displayed at will in any desired configuration on CRT screens, transformed into printouts and retransmitted, perhaps using one's own home computer linked by telephone lines to a receiving word processor or similar device in the editorial office of a journal?

2. PRESENT RULES OF CAPITALIZATION

To answer this question objectively, it is necessary to examine the rules on which present American editorial practices are based. Most American editors follow the rules of the *Manual of style* of the University of Chicago Press, the latest edition of which appeared in 1969 [3]. This code of practice essentially preserves rules of orthography and in particular those of capitalization that can be traced to the early days of the century and perhaps even further back in time. The relevant rule, 7.123 *Capitalization*, stipulates:

> Capitalize the first and last words and all nouns, pronouns, adjectives, verbs, adverbs, and subordinate conjunctions. Lowercase articles, coordinate conjunctions, and prepositions, regardless of length, unless they are the first or last words of the title or subtitle. Lowercase the *to* in infinitives.

The anonymous authors of this edict do not tell us (a) why any words other than first ones and proper names must be capitalized, just because they happen to be title words, contrary to normal English spelling rules; and (b) why articles, coordinate conjunctions and prepositions are being demoted to second-rate status, as it were. Indeed, nobody, not even the editors of the *Manual*, seems to know the answer, because in every American style book (and their number is legion) the same or similar rules are stated, but they are never justified as to the specific function they are to fulfill. It is just that the rule is on the books "because we have always been doing it this way".

As an aside, it should be noted that the dichotomy between coordinate and subordinate conjunctions by no means exhausts the several categories of conjunctions distinguished by traditional grammarians, such as copulative (which seems to be only a fancier version of coordinate, the standard example being the word *and*), correlative (*neither, nor*), adversative (*but, however*), illative (*therefore, thus*), and temporal (*when*), all of which are expected to be known and duly capitalized or not, as the case may be, by writers, editors, typesetters and proofreaders. The fact of the matter is that the classification of words in traditional grammars goes back to the eight classes of words listed in the *Technē grammatikē*, a Greek grammar written by Dionysius Thrax (1st c. B.C.) whose authority was never challenged until the advent of modern linguistics. There are no such rigid classifications of words, neither for Classical Greek nor for modern languages, much less are there "unimportant" words, because just the so-called "function words" (among which are the poor conjunctions and prepositions) are those parts of speech that make meaningful communication possible, especially in English which has shed almost all of its inflectional devices, and relies instead on syntactical position of words.

The author of the well-known *Manual for writers of term papers, theses and dissertations* [4] Kate L. Turabian, also of the University of Chicago Press, follows the same rules but adds some intriguing and even more arcane provisions, no doubt to enhance the "scholarly" flavor of term papers and theses, the contents of which often fall woefully short of being either scholarly or scientific. In her rule 4:5 we are told that in "scientific fields", capitalization of title words should be "kept to a minimum" (no reason given), but in the humanities and in "most of the social sciences" (no indication which of these do or do not qualify) the "rules given in the following paragraphs" are to be followed; those rules are a more elaborate recapitulization of those in the *Manual of style*. In the chapter on "Scientific papers", rule 12:9 shows two examples of citations in which title words are not capitalized.

That would seem to be in accordance with what was stated earlier in rule 4:5. However, in rule 12:17 we are told that the style of some additional examples follow that of "the leading publications in the several fields" (again, without naming them; only the author knows what, in her infallible opinion, are "leading publications", others need not know--they only have to accept meekly Turabian's pontifical dogma). The "fields" are exemplified by citations of works pertaining to anthropology, physiology, psychology, chemistry, mathematics, and physics; of these, it seems, chemistry and physics are not quite as scientific as the others, because the examples from these two fields *do* capitalize book titles, whereas the rest do not. Turabian does not tell us what is *de rigeur* in, say, astronomy, geology, biology or other scientific fields, so writers of theses in those areas are left to fend for themselves.

The only American Standard that addresses the question, ANSI Z39.29-1977, *American National Standard for bibliographic references* [5], is not unambiguous. For titles of monographs and serials its section 4.7.3 *Capitalization* gives the option of either capitalizing the first word of a title and each significant word thereafter (without specifying what is "significant"), or to capitalize only the first word and proper names; for "analytic-level titles" (which means articles, papers, etc.) however, only the second rule is to be followed. All of this is based on the provisions of the International Standard ISO 4-1972, *Documentation --International code for the abbreviation of titles of periodicals* [6].

Finally, the country's central cataloging agency, the Library of Congress, invariably uses regular English spelling for titles of monographs and serials in its catalogs, a practice automatically followed by all libraries as well as by the bibliographic databases.

Thus, the picture of chaos in the rendering of titles in bibliographic references as practiced in the United States is complete. Here, it must be mentioned that most British printers and editors long since abandoned special rules for capitalization of titles which appear both as headings of articles and in references and footnotes in normal spelling. Needless to say, this introduces yet another element of diversity into the practices of bibliographic citation.

3. DISPLAY OF TITLES IN DATABASES

The display of bibliographic data in hard copy by various database producers also follows different paths. Here, we encounter another dichotomy, namely the display of titles in normal English orthography, i.e. upper- and lowercase letters, as opposed to display in ALL CAPITALS. A brief check of some of the major databases revealed the following picture: *Biological Abstracts, Chemical Abstracts, Index Medicus, Public Affairs Information Service*, and all H.W. Wilson indexes (the latter available only in printed form) present titles in normal orthography; *Engineering Index, ERIC Resources in Education,* and all ISI services print titles in ALL CAPITALS. Needless to say, a title printed in the latter style must be entirely restructured before it becomes acceptable to an editor of a journal, regardless of its capitalization practices. For this reason alone, database producers should abandon the practice of printing titles in ALL CAPITALS. Other weighty arguments against this practice are the following: (a) research into legibility has shown that anything written in ALL CAPS is more difficult and time-consuming to read than text in upper- and lowercase [7]; (b) titles often contain abbreviations or acronyms which, if all are written in uppercase, tend to "disappear" among the words of a title or may be misread, e.g. IN may be the word "in", the symbol for Indium, the zip code for Indiana, or the abbreviation of *inch*; (c) the meaning of a title may be affected by obliterating the distinction between common words and proper names, e.g. "The restoration of old Egyptian mummies" is not the same as "The restoration of Old Egyptian mummies", and "Simple basic data files" does not mean the same as "Simple BASIC data files"; (d) chemical elements cannot be properly identified by their symbols, e.g. Co is not the same as CO; (e) diacritical marks used in the writing systems of foreign languages cannot be shown when ALL CAPS is being used, while it is at least possible (though normally neglected by American printers) to provide these important marks which may affect the meaning of a word, e.g. the German *Mohren* and *Möhren* mean Moors and carrots, respectively, and the Swedish *får* and *far* mean sheep and father. Although the context will often make it clear what is meant, there is still room for ambiguity and error.

If typographical diversity in the hard copy presentation of bibliographic data is desired, the names of authors may be printed in ALL CAPS (as is already the practice of some data-

bases and abstracting and indexing services). This is much less subject to possible error and misinterpretation than the complete capitalization of titles, although the problem of diacritical marks in personal names and acronyms as parts of corporate names still remains. Thus,wholesale capitalization should be avoided as far as possible, and different typefaces should be used instead, where feasible.

4. CONCLUSION

At a time when interchangeability and compatibility of bibliographic data are more important than ever because of the ease with which data can be transmitted electronically from an originator to various different storage media, and from any point on the globe to any other point--do we really have to submit to the chaotic conditions outlined here, imposed on the community of authors by unknown persons who arrogated to themselves the prerogative to dictate which English words, when appearing in a title,should or should not be "dignified" by capitalization? The answer is clearly "no".

Of course, we cannot and should not tell printers and graphic designers how to display titles of articles in journals. What we should be concerned about is the rendering of such titles in bibliographic control tools such as bibliographies, catalogs, lists of references, footnotes, and any other secondary recording of titles, and the most effective way of transmitting and exchanging such data, once they have been captured in machine-readable form, without the need for any restructuring of their orthographical form.

I hope that I have shown that, far from being minor or trivial, the graphic rendering of titles in database output versus their display in printed form (that is, in journals, bibliographies and lists of references) is indeed a problem. In the past, before the advent of centralized storage of bibliographic data in databases, this may have been merely an innocuous game played by editors, but it is now becoming a nuisance and an obstacle to rapid communication of data. Fortunately, it is also a problem that is amenable to an easy solution, not requiring allocation of funds or costly changes in equipment. It just needs some common sense, a readiness to cooperate and adherence to an already existing and accepted standard. The gnomes of Chicago have played their games of irrational and idiosyncratic capitalization for too long. The time has come to terminate such games, the rules of which rely on the nebulous authority of "time-honored practices". Thus, editors should follow the normal rules of English orthography which are the commonly accepted standard of written communication, following the example of most of their colleagues on the other side of the Atlantic, as well as that of the Library of Congress. As to database producers, it seems that none or only a few of them are following the archaic practices of editors, although they do have at least two different typographical practices for hard copy printout. They should also adopt the standard of normal English orthography for titles presented in hard copy, to make such presentation uniform, standardized, and easily legible.

REFERENCES

1. Cathro, Warwick S. The upheaval in bibliographic exchange standards 1974-1984. *Australian Library Journal* 16 (May 1980): 59-68.
2. Dierickx, Harold. State of the art of standardization of bibliographic data elements. (In: *International access to aerospace information. Technical information panel's specialists' meeting held in Athens, Greece, 17-18 October 1979.* Neuilly-sur-Seine: AGARD, 1980, p. 4=1-4=7. (Conference proceedings no. 279 / AGARD))
3. *A manual of style.* Chicago: University of Chicago Press, 1969.
4. Turabian, Kate L. *A manual for writers of term papers, theses, and dissertations.* 4th ed. Chicago: University of Chicago Press, 1973.
5. *American National Standard for bibliographic references.* New York: American National Standards Institute, 1977. (ANSI Z39.29-1977)
6. *Documentation--International code for the abbreviation of titles of periodicals.* Geneva: International Organisation for Standardisation, 1972. (ISO 4-1972)
7. Paterson, D.G.; Tinker, M.A. Readability of newspaper headlines printed in capitals and lower case. *Journal of Applied Psychology* 30 (1946): 161-168.

DISCUSSION

Linda Tepp, Recorder

Cataloging Distribution Service, Library of Congress
Washington, D.C.

The speakers were joined by two discussants, Joel Lipkin, King Research, Inc., and Alan Wenberg, National Technical Information Service. Mr. Lipkin is a Senior Research Associate at King Research, with primary responsibility in the area of data processing. Among his more recent activities is heading the team working on the logical design for the Dept. of Energy's data resources directory. Mr. Wenberg is a staff member of the Office of Data Base Services at NTIS, where his group is responsible for identifying available software, accessioning and distributing machine-readable products, providing technical assistance to users, and operating the Federal Software Exchange Center for the General Services Administration. Linda Tepp, the recorder for this session, is a library information systems research analyst in the Cataloging Distribution Service of the Library of Congress. Formerly, she was a program analyst in the Copyright Office of the Library.

The discussion began with Ms. Dodd's paper in which she suggested that the user header labels could be a useful option to implement. Ms. Maruyama posed the question as to whether this would be a practical point to consider and whether this could become a FIPS standard. Mr. Wenberg said that as far as he was concerned personally, user header labels would be very useful. Overcoming user resistance, however, would be the biggest problem because in distributing software and data files, approximately five to ten percent of users ask NTIS to avoid labels.

As a historical footnote, Mr. Wellish commented that the first books produced during the forty to fifty years after the invention of printing did not have title pages. Such things did not exist because manuscripts did not have title pages. It took quite some time before the printers of that time realized that it might be useful to have some kind of general label or announcement up front that would tell readers, first the author, then the title, and even when and where the book was printed--information that was normally relegated to the end of a manuscript as it still is in Japanese books today. Since data bases, historically speaking, have just been born yesterday, it may also take a few decades until the producers of data bases or other such documentation begin to understand that the people who invented the printed book came up with some clever innovations, namely title pages. Ultimately, title pages in data bases may become commonplace.

An attendee from the Control Data Corporation said that user header labels sound like a very good idea as long as they don't interrupt the current labeling facility and they're carefully implemented so that they don't produce any more resistance. Since Mr. Wenberg has already mentioned some user resistance to labeling and it took the industry a long time to get where it is with labeling, he would prefer having what we have now rather than winding up with less.

Mrs. Butler said that the real problem with labels has always been that they weren't easily portable from one type of hardware to another. Also, the fact that some of the files are binary data files has created difficulties. She mentioned another issue that came up when standard labels were being debated. ASCII is the only character code that has a standard, so the user header label has to be an ASCII standard label. Many data files that are interchanged, however, are not in ASCII character code.

Ms. Dodd added that she was well aware of the problems involved in using this particular option and the resistance on the part of computer facilities, but somehow, the

spirit or the intent has to be manifested, possibly in another way. She raised the question as to whether this is the only option available to us now, and because it is being resisted and not being used, what could be an alternative? She asked if there exists some other kind of standard or labeling device or title page equivalent that could be incorporated in the file itself because it's extremely important to have this in terms of maintaining some kind of control or data base management. Mr. Wenberg suggested that one possibility might be to create a small file in front of the actual data file or program file that is being interchanged. It would be a separate file in front of the data file; it would not be the first record in the file. Ms. Dodd said that such a proposal had been made in the past and agreed that it probably deserves more attention.

Mrs. Butler brought up another point that a distinction should be made between tapes for tape-resident files and tapes for transmitting data. The latter is merely like mailing a parcel. Most people who are using the data files are not using them from tape but from another resident device. She also expressed a need for something interchangeable besides tape pretty soon.

The next major discussion point concerned FIPS 30 (Software Summary for Describing Computer Programs and Automated Data Systems). In response to a question from Ms. Maruyama, three attendees (two from the National Technical Information Service and one from Calculon Corporation) indicated that they are actively using FIPS 30.

The papers presented by Ms. Maruyama and Mrs. Butler addressed the need to revise FIPS 30 to meet changing bibliographic requirements and to reflect the new ANSI X3.88 standard for computer program abstracts. Mr. Lipkin suggested that before deciding what should be included in FIPS 30, a question should be asked as to why you want to obtain someone else's software. In addition to putting the software up on your machine or using their data base management system, there may be some other things that you would want to do with someone else's software, which might necessitate describing attributes that had not been considered up to this point. For instance, you might want to take someone's module and look at it in terms of an algorithm and implement a certain capability. You might want to look at a system design as implemented. Or, you might want to copy a piece of the software or modify an existing system. These factors might require different kinds of material as part of the software documention. Perhaps we've been talking mainly about whole systems, such as transporting a circulation system or a data base management system, but the possibility for sharing individual modules does exist although this area has not been addressed at any great length.

In answer to Ms. Maruyama's question as to whether this would be appropriate for FIPS 30, which consists of the program abstract, as opposed to FIPS 38, which is the software documentation itself, Mr. Lipkin said this probably relates much more to FIPS 38. ADA as a language, for example, provides this sort of capability for documenting packages--small modules which could be considered a library of data packages that could be shared. Mrs. Butler added that FIPS 30 does include the capability to describe modules as well as entire systems.

The next discussion point considered whether a standard comparable to FIPS 30 should be developed for machine-readable data files. Ms. Maruyama mentioned in her paper that the Office of Management and Budget had issued a directive in an attempt to standardize some of the elements related to statistical data files. Although certain agencies are mandated to follow this directive, it doesn't have quite the same clout as the FIPS standards.

Ms. Dodd said that the directive is very close to what Mrs. Butler had described in her paper for software. The elements in the directive start with the bibliographic citation for the data file and then go into identification number, summary, geographic and time coverage, technical characteristics, and so on. The standard should not be limited to computer software, so it would be a good idea to extend it to other types of files. She recommended that such an action be considered.

An attendee questioned whether the Paperwork Reduction Act doesn't mandate a certain degree of this. With far greater usage and sharing of government data, how will the establishment of standards be affected. Is the Act already affecting standards? Mrs. Henderson mentioned in this context that a federal information locator system (FILS) is being established which is supposed to be a file of descriptions of data collections--not a description of the data themselves but the existence of a data collection. She hoped that the people who are developing FILS are adhering to FIPS standards if they apply.

Mr. Lipkin said that one of the preliminary reports on FILS was devoted to the subject of their own data element element dictionary for describing government data files, so he surmised that they're developing their own data element dictionary. In effect, Mrs. Henderson continued, they're trying to compile a dictionary of all data elements found in all of the files so that one would be able to find a particular data element in certain files across the whole government. Mr. Lipkin also mentioned that FILS was not limited to machine-readable data files but was focused mainly on government reporting requirements. A related project attempts to do the same thing specifically for machine-readable data.

The last discussion topic involved Mrs. Henderson's paper on the possibility of developing guidelines or standards for compiling data element dictionaries for bibliographic information. An attendee from the Dept. of Energy's Technical Information Center said that the discussion, thus far, has concentrated on data elements, standards for data elements, and incorporating ANSI X3.88 into FIPS 30, all of which seem to be applicable for computer software and perhaps for numeric data. He asked Mrs. Henderson if she was referring to a data element dictionary that merges bibliographic data and software.

In response to this question, Mrs. Henderson said that in talking about a data element dictionary, she attempted to relate that effort to data directory systems (DDS), for which the Institute for Computer Sciences and Technology has generated guidelines and proposed a FIPS standard. One of her former colleagues reminded her that software documentation or program documentation is a different problem and is handled by a different part of the institute from systems documentation where the DDS belongs. She still contended, however, that enhancing software exchange can also include the data that the software carries. As such, it is a management tool that is part of management's total arsenal of tools to improve, maintain, or make more effective its computer-based systems, particularly for bibliographic data processing.

The audience may have been misled by her mentioning the study of interaction among the four computer systems of the four major reports-processing agencies as the first step toward sharing data. The data element dictionary project could help to determine what data ought to be shared.

The attendee from the Dept. of Energy said that his interest in this matter stemmed from his attempts to move NTIS from a data element concept for bibliographic data to a similar approach for software. In this context, Mr. Lipkin reported that several of the data dictionary packages available now allow you to record information about program modules, or systems, as well as about files and record interrelationships among numeric and bibliographic files, and the software processes. From his perspective, this is one of the really important advances because it allows you to maintain documentation in synchronization with maintenance and development information that might be collected in the process of changing a module and updating the dictionary record. That's one method to keep your documentation current. In response to a comment that there might be confusion here between bibliographic data elements and software data elements, Mr. Lipkin said that although they are different, they do interact. If you change your files, you will also want to be certain that the programs that process the files are recompiled. The integrated dictionaries provide the capability to monitor the fact that you have recompiled your program before you try to run it against the file. It forces you to good practices.

Mr. Lipkin also summarized the project that he is presently working on, which is somewhat related to the FILS project. For the Energy Information Administration (EIA) of

the Dept. of Energy, a data resources directory is being developed which is, initially, a description of all the data collection systems within the department--about 160 systems. The forms are described in the context of bibliographic terms and data collection frames. Data elements within the management system are also included, and enforcement for any modifications is handled by the fact that the same system is used by EIA to get forms approved by the Office of Management and Budget (OMB). In other words, if a new form is desired, that information is input into the system and output as a report for OMB. The system also allows managers to find out what information they could collect on these forms. The system is being expanded to machine-readable data files and to publications in an attempt to create a linkage between the two "files" and to include descriptions of major software packages which access the files. What is being attempted is to develop an environment where someone can manage EIA as a data processing system, consisting of over 160 systems, over 100 publications, and thousands and thousands of machine-readable data files.

SUMMARY

Lenore S. Maruyama, Moderator

Network Development Office, Library of Congress
Washington, D.C.

As seen from the Discussion portion of this session, there was considerable dialog among all the participants (speakers, discussants, audience) on the papers presented, and it is hoped that the exposure to the different aspects of software sharing has stimulated thinking in this area. The conclusions listed below, however, include only those that might lead to further action by the National Bureau of Standards in the near future:

- Although there was general agreement on the usefulness of a user header label to record bibliographic information describing a machine-readable data file, the overall discussion indicated that there may be too many difficulties (some perceived but many real) to overcome that might prevent its acceptance and implementation. One alternative suggested was to create a separate (small) file in front of the data file to carry this information instead of carrying a user header label.

- The consensus of the group was that FIPS 30 (Software Summary for Describing Computer Programs and Automated Data Systems) should incorporate the new elements from the American National Standard for Computer Program Abstracts (ANSI X3.88).

- FIPS 30 should be expanded to cover numeric data files, or a new FIPS should be compiled to handle these files. This work should be based on the directive issued by the Office of Management and Budget concerning statistical data files.

SESSION G: Improving Human Interfaces

Moderator: Joseph Psotka, National Institute of Education

Users of software documentation face many practical problems that can be overcome by a proper design of the human interface. This session will deal with several interface characteristics designed to make documentation friendlier, more useful, and of higher quality.

Designing Software Documentation for Non-technical Users

V. Douglas Hines

House Information Systems
U. S. House of Representatives

FIPS PUB 38 provides guidelines for "automated systems" documentation, which includes "User Manuals." The target audience for this documentation -- "non-ADP professionals" -- represents an increasing proportion of software users. Documentation for this group should contain the basic information described in FIPS PUB 38, and should meet broad standards of clarity, completeness, accuracy, and ease-of-use. In addition, five specific design techniques are recommended: (1) emphasize procedures for using the software; (2) organize documentation by system function; (3) provide specific and complete examples; (4) develop documentation in two phases, with programmers writing initial version to be modified by user support specialists; and (5) incorporate documentation into user training.

key words: automated data systems, user manuals

1. INTRODUCTION

Historically, software documentation has been written primarily by programmers for programmers. It has involved one programmer telling other programmers what his or her software did, how it did it, and why it did it that way.

With the widening use of on-line data bases, electronic messaging (mail) systems, and other office automation applications, the population of software users has broadened to include many individuals who are not full-time data processing professionals: managers, accountants, engineers, office workers, and a host of other professions. Each year, these users increase in number and become a more important audience for software documentation.

Hence, "user manuals" -- that class of documentation in which programmers talk, not to other programmers, but to

non-technical software users -- become progressively more imporant in many ADP environments. From the point of view of the software developer, this documentation is a key determinant of the acceptance and usefulness of the software.

The U.S. House of Representatives is a data processing environment in which software documentation for non-technical users is particularly important. HOUSE INFORMATION SYSTEMS (H.I.S.), the computer service group for the House, supports 9 on-line data bases, 2 scheduling systems that require data entry, an electronic mail system, a text processing system, and a variety of more specialized systems. There are about 3000 accesses to H.I.S. computer systems distributed to users in Washington DC and across the country; the vast majority of these users are not data processing professionals.

Under these conditions, the quality of user-oriented software documentation is a crucial consideration, and consequently significant attention has been paid to its development. This paper describes a set of simple design principles which have evolved as H.I.S. has attempted to meet the challange of producing high quality user-oriented documentation.

2. RECOMMENDATIONS FOR CONTENT

Of the content areas for User Manuals outlined in FIPS PUB 38, "procedures and requirements" is of greatest interest to non-technical users. Experience has shown that non-ADP professionals have only passing interest in "general information" about the software and technical characteristics of the "application." Moreover, when dealing with a geographically dispersed user population, information about performance and equipment is difficult to summarize in a form that will hold the interest of non-programmers.

2.1 RECOMMENDATION ONE: The content of user manuals should emphasize procedures and requirements.

Our first design principle is that the content of user manuals should focus primarily upon procedures, i.e., how to use the software. General information about data base structure and other features of the software should br provided. However, non-ADP professionals are primarily interested in using the system, and the bulk of the documentation should focus upon telling them how to do so.

3. RECOMMENDATIONS FOR FORMAT

There is general agreement that all software documentation should be clearly written, easy to use, accurate, and complete. As most of us are painfully aware,

much of it isn't. But even if software documentation is clear enough to meet the needs of programmers, it still might not meet the needs of non-programmers who are not accustomed to working with computer systems.

Documentation is more understandable to non-programmers when the software is described as the user sees it -- as a set of functions that are performed in accordance with a predetermined set of system commands. The key to good user manuals, then, is to (1) list and describe the functions, and (2) list and provide examples of the commands or responses that cause the functions to be performed. In short, the documentation describes what the software does and how it does it. Hence, our second and third design rules.

3.1 RECOMMENDATION TWO: Documentation should be organized by function as understood by the user.

Software documentation is usually organized in terms of commands or other program design parameters; this is reflected in tables of contents that list commands and page references, without telling the reader what the commands accomplish functionally.

Instead, it is recommended that documentation be organized by function, making it as clear as possible to the user what the system can do and the commands necessary to do it. For example, in systems using STAIRS software print specifications are set under the browse command. It is more meaningful to have a section in the documentation entitled "setting print specifications" than to have one entitled "the browse command." Whenever possible section headings and the table of contents should reflect system functions that the user understands.

Organizing documentation by function has added benefits in environments such as ours where a non-technical user might work with five or six different software packages. It turns out that the basic functions of on-line data bases are surprisingly consistent. Users sign-on, select data base subsets, set output formats, perform searches, print or save queries, then change data base subsets or sign-off. There is more variety across text and word processing systems, but a small number of generic functions can be identified here as well. Users create a document, set format (tabs, margins, page length, etc.), enter and edit text (involving cursor movement and edit commands), print and store documents.

When user manuals are organized by function, there is more comparability across documentation and more transfer of learning from one on-line system to the next or from one word processor to the next. This makes each system and manual more valuable.

3.2. RECOMMENDATION THREE: Provide at least one specific example of each system function or command, showing both the "user" and "system" responses.

FIPS PUB 38 reflects the fact that "input formats" are the key element of the procedures and requirments of automated data systems. In on-line data bases and other interactive systems, software usually allows for a system prompt, to be followed by a user entry (usually a command or parameter), followed by a system response (usually output data answering the user query). Documentation should provide specific examples of how this sequence would look for each major system feature.

This approach makes it easy to provide users with the information discussed in PUB 38: input length, sequence, vocabulary, punctuation, etc.

4. PROCEDURAL RECOMMENDATIONS

The final recommendations concern procedures for creating and distributing documentation.

4.1 RECOMMENDATION FOUR: Develop software documentation in two phases.

Documents should be developed in two phases. First, the applications programmers and analysts who write the software create a technically-oriented, detailed, and complete guide to the software. This document includes system design philosophy, a description of data base structure and parameters (if applicable), and listings of special commands, features, default options, etc.

In the second phase, user support personnel (such as training specialists) take this document as a starting-point for the software documentation targeted to the end-user. This second-iteration produces the user manual that is released to non-ADP users. H.I.S. refers to these as "how-to manuals."

4.2 RECCOMENDATION FIVE: Use the documentation during training.

Most software packages designed for non-technical users are supported with training, often in a formal classroom context. Frequently, software documentation is handed out at the end of these sessions or, if distributed earlier, is not refered to during training.

A better approach is to distribute the documentation at the beginning of training and use it as the basic reference;

this way, users learn to use the documentation as well as the system itself. In other words, classes teach both how to use the software and how to use the documentation. This seemingly trivial point has enormous practical implications for increasing the sophistication of inexperienced software users.

5. CONCLUSIONS

At one point in the evolution of data processing, software documentation was written by programmers for programmers. Today many large installations and software vendors who create software used by non-programmers are encountering new audiences and are discovering the importance of the once humble user manual. This paper has recommended ways to design this type of documentation so as to maximize its contribution to the success of a software package.

Paper and Glass: Graphic Design Issues for Software Documentation

Aaron Marcus, Staff Scientist

Computer Science and Mathematics Department
Lawrence Berkeley Laboratory, 50B-3238
University of California
Berkeley, California 94720

Graphic design principles have been utilized in redesigning the interface for an information management system and for prototypes of typographicly enhanced textual programs. These principles are explained and examples of typical formats are shown to indicate the nature of improvements.

1. Introduction

Most programs and their supporting documentation pass through many stages of development, use, and maintenance. These software documents may appear offline on paper or video. They may also appear online displayed on a paper or glass-faced terminal. These documents communicate their contents to the reader primarily through alphanumeric symbols. These pages or screens of information must effectively communicate intentions, states, structures, and processes. While good conceptual organization and verbal editing are crucial to effective communication, a third component, the graphic design of these documents, has been neglected.

Graphic design is the discipline concerned with the communication of informational, emotional, and aesthetic content through the manipulation of typography, symbolism, illustration, color, spatial organization, and temporal sequencing. [1] Certain professionals in this disipline are concerned primarily with the communication of complex information through the design of charts, maps, diagrams, and other technical documents. Knowledge from these professionals and their literature can be applied to the task of designing the graphic presentation of software documentation which now faces builders, users, and managers of computer systems. Graphic designers usually are not involved in setting up conventions, standards, and specifications for producing software documentation. In order to educate the information specialists and computer scientists who normally rely upon their own limited expertise, this article focuses on the typographic principles of information oriented book or document design drawn from the professional literature and from the author's own experience as a graphic designer of computer-based documents [1;2;4].

The software documentation interface between the human being and machine is in the context of the person using a computer system and in the person building or maintaining a computer system. Elsewhere the author has termed these the inter-faces and the inner-faces of computer systems. [2] Basic principles of selecting visual signs and their attributes (such as their location, size, and boldness) for presentation on both paper and glass can enhance the legibility of software documentation as well as its readability, i.e., its appeal or friendliness.

2. Typographic Aspects of Graphic Design

The design task concerns determining a relatively high degree of fit among the different requirements of the components of every communication interface:

the sender (the machine or user)
the medium (the display device)
the receiver (the user or machine)
the message (the information content).

By means of the position, color, size, grouping, and temporal sequence of visual signs such as alphanumerics and symbols, the graphic designer must convey the usual facets of a software documentation system: continuous prose (e.g., help messages and lengthy explanations), interrupted prose (e.g., error messages, system status reports, examples), and tables or lists (e.g., source code, menus, data dictionaries).

Typographic design begins with a concern for the design of individual symbols. In many current display systems there is relatively little control over symbol design. A limited hardware set of characters is often used to display alphanumerics and other symbols. Because many terminals and printers currently operate with fixed-width characters, many of the principles given below are oriented toward this situation.

In online display, there is often little control over symbol design; it is likely that the standard medium for interaction may be a display showing 24 lines of 80 alphanumeric characters each. The use of reverse video, italic, or levels of brightness can not always be assumed. Even if these means of visual emphasis are not used, other approaches are available. For example, there can be a strong reliance on a horizontal line of hyphens to highlight certain titles or to separate divisions of the frame.

Even within severe limitations, attention to graphic design principles can improving the effectivenes of software documentation. Consider the use of all upper case words, a typographic approach which much documentation utilizes. The fixed width of the letters are often created by a 7x9 or similar dot matrix. In such conditions lower case letters with occasional capitals are more legible. Research shows [3, 35] that not being able to perceive word shapes (as is true for words set in upper case characters only) may slow reading speed by as much as 13%. Because line printers and terminals often have little space between lines in comparison to normal textbook typography, lower case letters are particularly important in providing visual space between lines of type and thereby improve legibility. In interactive situations, lowercase typography for machine messages and for the echoes of user input should be used whenever possible. When all capital settings are used, they should be used to highlight a restricted set of pri-

mary content elements, e.g., the main title of a frame or the module in which a prompt occurs.

3. The Grid

As for the design of a traditional printed book page, the graphic designer of software documentation must consider the visual field, the terminal screen or the printed page, as an entity whose proportion, size, and distance from the viewer are important to the design of information. Information is presented in conceptual frames of pages or screens. To assist the overall organization of elements within the frame and consistency from frame to frame, a reference grid of a few horizontal and vertical lines should be determined to locate certain standard positions for elements such as titles, prompts, etc. One of the most important functions of the grid is to establish certain basic divisions of the frame. The grid should establish one or more major columns of text of approximately 60 characters in width.

For fixed-width character printers or terminals, one simple approach to frame design is to use two primary locations: a single major column lying between character positions 21 to 80 and a special position at character position 1 for all secondary matter, such as subtitles for explanatory text or user input for textually oriented command and control interfaces. For subtitling, the reader can easily scan the overall structure of the document; for interfaces, the user's input and the machine's responses are visually distinct. Primary tab settings of 10 characters each and a secondary set every 5 characters can help divide the entire visual field into regular, modular units. Selection of upper or lower case alphanumeric characters and a grid influence other aspects of the typographic design, viz., character spacing, word spacing, line length, justification, line spacing, and the overall spatial structure of the frame.

4. Words, Lines, and Paragraphs

In situations in which character width is constant and letterform design is quite simple, word spaces are relatively large and lines of text tend to fall apart into a loose collection of alphanumerics. Wherever possible the typographic design approach stresses the need to keep words that belong together close to each other in word, line, and paragraph groupings. For example, only one word space is sufficient after a period in continuous prose to separate the end of one sentence and the beginning of the next. The graphic design approach also seeks to emphasize clear spatial groupings over the entire visual field in order to make distinctions of content. At the same time these spatial groupings are limited in their variation so that there is an overall visual consistency or rhythm within and between frames.

A typical oversight in most textual displays is using text lines of too great a width. Normally there should be approximately 40-60 characters per text line (about 10-12 words) [3, 29]. Research has shown [3, 33] that unjustified (unequal length) text lines are just as legible as justified text. In the case of fixed width characters, justification usually means that large gaps of empty space appear between words in order to achieve equal width text lines. These large spaces interrupt eye movement and impede reading. Especially for interrupted text, typographic design calls for unjustified paragraphs. This design feature has the added effect of making character position 21 visually the most important in the frame. An implied vertical line of the beginnings of text lines appears at this position. This becomes the location for many key words, text line beginnings, etc. The reader quickly develops the habit of scanning this location for most beginnings of information.

In fixed character width, fixed interline spacing situations, the space between groups of lines has limited variation. Whenever possible one should avoid any spacing larger that a single line skip. This may be used between paragraphs, line clusters, individual sets of menu prompts, user responses, etc. In this way a maximum number of text lines per frame can be utilized. Note that the horizontal line made of hyphens can replace a skipped text line and does not add another line to the already limited number of lines in a frame.

5. Tables and Lists

A major design principle is to limit the amount of variation wherever possible. This applies especially to tabular settings for tables and lists. In the case of fixed character-width situations, the most important words or word groupings are placed at or near (i.e., before or after) the tab setting at the 21st character position. All tables and lists require headings to describe the contents in general and to identify the parts if there are many. These titles should not scroll off the screen or disappear from continued pages; they should be regenerated as needed so that each frame includes sufficient titles to be comprehendable. All horizontal positioning of tables and lists is governed by the desire to keep codes, page numbers or other symbol groups close to the items to which they refer and to allow easy scanning down and across items.

6. Examples

The principles outlined above are embodied in two sets of accompanying examples. One set involves redesigned formats for the low resolution online interface [4] to an information management network which accesses very large geographic databases [5]. The other set arises from prototype redesigns of textual programs for display on high resolution terminals or printers. A comparison between old and new versions will clarify how earlier designs for frames were faulty and inconsistent. Improvements in the newer versions should be obvious. The examples appear in the accompanying Figures.

7. Conclusions

Most of the changes in the documentation formats have been relatively easy to implement within the software. These redesign features are more than a 'cosmetic' facelift to the system. By carefully considering not only what to show, but also when, how, and why to show it, a better understanding of the functionality of the system emerges in the minds of the builders and ultimately in the minds of the users of the computer system.

Many of the changes in design constitute working conventions rather than carefully proven standards. However, in the case of the first set of examples, many of

the changes corresponded to recommendations of an independent critique of the system [6, 54-55]. In the second set it is also clear from informal discusslons with users and implementors of computer systems that changes brought about by consideration of typographic design principles have made clear improvements that programmers as well as users can readily perceive. As these design principles and specifications for new documentation standards are more completely determined, they can be embodied in a graphic design manual [7]. This manual could assist future builders of documentation modules to maintain a consistent, high quality inter-face or inner-face for the computer system.

Acknowledgements
This work was supported by the Applied Mathematical Sciences Research Program of the Office of Energy Research, U.S. Department of Energy, under contract W-7405-ENG-48. The author also wishes to acknowledge Dr. Ronald Baecker, President, Human Computing Resources, Toronto, with whom the author developed the textual program visualization and Mr. Richard Sniderman of Human Computing Resources, who helped write some of the typesettlng macros for that visualization.

References

1. Marcus, Aaron, "Computer-Assisted Chart Making From the Graphic Designer's Perspective," *Computer Graphics*, 14:2, 1980, 247-253.

2. Marcus, Aaron, "Graphic Design and Computer Graphics", *Industrial Design*, March/April 1982, In press.

3. Rehe, Rolf F., *Typography: How to Make it Most Legible*, Design Research International, Carmel, Indiana, 1974.

4. Marcus, Aaron, "Designing the Face of An Interface," *IEEE Computer Graphics and Applications* 2:1, January 1982, 23-26ff.

5. McCarthy, John, et. al., "The Seedis Project: A Summary Overview", Publication 424, Lawrence Berkeley Laboratory, September 1981.

6. Bleser, Terry, Peggy Chan, and Mei Chu, "A Critique of the SEEDIS User Interface," Report GWU-IIST-81-04, Department of Electrical Engineering and Computer Science, The George Washington University, Washington, D.C., March 1981.

7. Marcus, Aaron, "A Graphic Design Manual for Seedis", in preparation.

Figure 1a: Undesigned Command Menu Descriptions Frame
Within the Computer Science and Mathematics Department of Lawrence Berkeley Laboratory, the author (who has a professional background in graphic design) has begun to apply the principles of information-oriented typographic design to the redesign of the interface for a large experimental geographic information management system called Seedis [5]. The interface for Seedis has gone through several stages since its genesis as a series of stand-alone batch programs in 1972, particularly as it expanded its functional capabilities. The current version of Seedis operates in an interactive VAX/VMS environement with a textual (i.e., essentially alphanumeric) interface. Seedis permits a relatively computer-naive person to examine data dictionaries, extract data from databases, to aggregate or disaggregate data between different levels of detail, and to display the selected data as a labeled table, dot matrix chart, pie chart, line chart, bar chart, or area/symbol choropleth map. In the Figure, note the illegibility of all capitals in comparison to upper and lower case and the interrupted list of command definitions.

Figure 1b: Designed Command Menu Descriptions Frame
The command menu description frame appears when the user types a question mark at any decision point, i.e., if there is some confusion about the proper response to the immediately preceding prompt. Note the organized appearance of text groups, the order of text elements, the use of rules, lower case, and specific tab settings. The full screen width is equivalent to 80 typewritten characters in width. Information on global commands is introduced in the very first information to the user. The standard form of the menu-prompt identifies the module (all capital letters) in which the user is currently working and the appropriate commands at this point. Note the use of the standard tab settings at position 1 and 21 and the consistent use of standardized verbs to describe the input commands. Global commands are separated from local commands appropriate to the particular decision point. The list is labeled to aid identification of its component parts.

```
?
 TYPE ONE OF THE FOLLOWING COMMANDS...
 ?             FOR THIS LIST OF COMMANDS
 HELP          FOR HOW TO GET HELP
 MORE          TO SEE NEXT SCREENFULL
 TABLE         FOR THE TABLE OF CONTENTS
 <N>           FOR PAGE <N>
 * <COMMENT>   TO ENTER A COMMENT IN THE LOG
 DATA <SEQUENCE LETTERS>         SELECT DATA CODES
 CANCEL <SEQUENCE LETTERS>       CANCEL DATA CODES
 FOR X <C>     SUBSTITUTE C FOR X IN DATA CODES -
               ALSO XX XXX XXXX Y YY YYY YYYY
 REVIEW        LIST DATA SELECTIONS MADE SO FAR
 SAVE          SAVE DATA SELECTIONS AND RETURN
 QUIT          CANCEL DATA SELECTIONS AND RETURN
 READY
```

```
                    DATA: <line letter(s)>, table, <page number>, CR
: ?

Input               Description
------------------- ------------------------------------------------------------
<line letter(s)>    select one or more data elements by line letter
table               display table of contents for this database code
<page number>       display a particular page
CR                  (carriage return) display the next page

?                   list avalable commands in this menu
help                describe data element selection
show                display table of contents for this database
review              list current data element selections and history
cancel              delete current data element selections for this database
quit                return to database selection menu

                    DATA: <line letter(s)>, table, <page number>, CR
:
```

```
READY
  MONITOR.SEEDIS.HELP.

                    INTRODUCTION TO SEEDIS

 The three major processes in SEEDIS are:

 AREA: define a geographic study area (composed of states,
     counties, or census tracts)

 DATA: select data appropriate to the geographic study area
     chosen.  For  example, for a study area consisting of a group
     of states, only state level data, and  not  county  or  tract
     level data, are appropriate.

 DISPLAY: manipulate and display the data in table, chart, graph,
     and/or map form.

 Normally  AREA,  DATA,  and  DISPLAY  are  performed in the order
 given. However, once the geographic study area is defined (AREA),
 one  may  alternate  between  DISPLAY  and  the   selection   and
 extraction of additional items in DATA.
TYPE MORE TO SEE NEXT SCREENFULL
TYPE ? FOR A LIST OF COMMANDS
```

Figure 2a: Undesigned Help Messages Frame
Note the long lines of text, the clutter in the last paragraph caused by clumps of all capital words, the gaps in word spacing caused by justification, and the mixture of small indentations with centered headlines.

```
                    SEEDIS: area, data, display, profile
: help
                    USING SEEDIS
                    ------------------------------------------------------------
                    LBL's Seedis is an experimental information system that
                    includes integrated program modules for retrieving, analy-
                    zing, and displaying selected portions of geographically
                    linked databases. Program modules in Seedis include:

          area      select geographic area (level and scope of analysis)
          data      select, extract, enter, or transform data
          display   manipulate and display data in tables, maps, and charts
          profile   produce standard socio-economic reports for selected areas

                    Normally Area, Data, and Display are used in the order
                    given. However, once the geographic study area is defined
                    in Area, you may alternate between Display and
                    the selection, extraction, or entering of additional items
                    in Data.

                    SEEDIS: area, data, display, profile
```

Figure 2b: Designed Help Messages Frame
Help messages are a standard one frame page description. Note the use of standard tab settings, unjustified text, the use of all capital headline together with hyphen line, removal of all capital keywords (replaced by exdented words, i.e., positional emphasis), and the use of second person in English language style. Further frames of information are available on the four key words listed.

Figure 3a: Undesigned Textual Program
This figure presents a typical C program in an elementary typographic form using fixed-width characters of a single size and typeface with limited horizontal spacing variation. There is little typographic hierachy. The program is more readable than those presentations that use all-capital typography and multiple commands per line, but there are still ways in which it can be made more readable.

Figure 3b: Designed Textual Program
This figure shows a prototypical black-and-white visualization that would require a high resolution bit map display terminal or a very high resolution hardcopy device. The actual image was generated in Times Roman type using a computer-controlled phototypesetter, a rare but not unheard of hardcopy device. This image is one of a series of experimental prototype frames for offline or online documentation that illustrates the full potential of a graphic design aproach to textual program visualization. The image was designed by the author and Dr. Ronald Baecker with Mr. Richard Sniderman of Human Computing Resources Corporation. Spatial location, typographic symbol hierarchies, figure-field enhancements, indexes, abstracts, etc., are combined to create a clear, consistent, explicitly structured frame that is legible and appeaing to the reader, based on a limited number of disussions with programmers who have viewed but not used this presentation.

Quality Issues in On-line Documentation

Joseph Psotka, Ph.D.

National Institute of Education
1200 19th St. N. W.
Washington, D. C. 20208

With the increasing use of microcomputers in all areas, the computer is taking on aspects of an appliance that only needs instructions to set up and then starts to work. This makes on-line documentation an important area for research. Suggestions are made here for the human factors aspects of on-line documentation, including a metaphor for guiding novice users; abbreviated menus for expert users; error messages that are polite; HELP statements that do not erase the current display; interfaces that respond to natural language statements; and input-output devices that make use of general skills. Although this view of documentation exceeds traditional print perspectives, it may be necessary to see documentation as part of the structure of a program when it goes on-line. These issues are discussed within the context of an educational software authoring system.

Keywords: On-line documentation, Human Interface, Authoring

1. INTRODUCTION

On-line documentation is becoming increasingly possible and increasingly necessary. It is becoming possible because microcomputers are appearing as dedicated machines that are used primarily for one function rather than as general purpose computers. As Heines (1981) has phrased it, the personal computer or microcomputer has begun to take on the distinctive characteristics of an appliance. This means, in part, that the users of the computer are much more naive about computers than traditional users, and they may have attitudes shaped by the use of other appliances, like refrigerators and television sets, that lead them to expect it to work in certain ways. In particular, they may expect (and perhaps deserve) to have the training and documentation incorporated right into the running of the system. This raises human engineering problems whose solution are only vaguely visible at this time.

Note: The views and opinions in this paper are those of the author, expressed in a personal capacity, and do not necessarily reflect the policy of the National Institute of Education, Office of Educational Research and Improvement.

This article is structured by using a functional analysis that examines the needs of an authoring language from the perspective of its three major users: the author(s), the teacher, and the student. Each of these users has their own set of needs in using the computer, but the needs all combine in the sense that training and documentation must be made a part of the system, if the authoring language is really to be useful. By examining how documentation might be made part of a educational software authoring system in detail, general characteristics of on-line documentation may become more apparent. Some of these general issues are raised in the first, overview section.

2. GENERAL ISSUES

2.1. Overview

Writing software for education (usually called courseware) is a highly labor intensive enterprise, like teaching itself. An authoring language that would allow teachers to write the programs is needed to reduce courseware costs, increase quality, and widen the range of materials available. The authoring system should also allow the software to be run on a variety of machines (increasing transportability). Therefore it needs to be used by teachers who want to use others' courseware. Finally, it should interact with the student to help him use the courseware written by the teachers. Thus, the authoring system needs to deal with many novice users who may have no detailed conception of what a computer is, nor what its limitations are.

2.2 Overall Metaphor

Novice users will come to the system with expectations that derive from their past experience in doing similar things. These intuitions need to be used carefully to make the system easier to learn and document. For the authors, a guiding metaphor might be developed that used their experience in writing a book, with its various activities of gathering material, writing content, illustrating, editing, publishing, proofing, and rewriting. For the teacher, the appropriate metaphor might be to see the authoring system as a novice teacher that needs help and supervision. For the student, the appropriate metaphor might be to see the authoring language as a teacher's aid, to provide audiovisual displays, adjust the lights, and provide additional exercises and explanatory material. Each user should have a metaphor that makes it easier to learn the system.

2.3 HELP Access

HELP functions are common to many systems, but they usually are quite subordinate to separate documentation that really describes how the system works. If documentation is really to be made on-line, then HELP functions must be reconceptualized to provide more detailed and useful information to novices users who have not read external documentation. Aside from general orientation metaphors and descriptions, and prompts about what to do next, the HELP functions may also need to simulate certain activities so that the novice can step through the function and obtain information about how the process should be conducted in some optimal way.

Current approaches to help functions have some severe shortcomings. One general feature of HELP functions I have used is that they require addressing a menu or a directory that forces one to leave the active display or workspace. A novice, in particular, cannot be expected to remember the exact characteristics of his work leading up to the problem. Even experienced users may have this problem. At the very least a windowing technique should be used to leave the display more or less intact while displaying the HELP information as well. This feature is even more important when the HELP function has to simulate a particular activity while the user is actually engaged in that activity. With a split screen or window that provided

carefully prepared illustrations of a function, while the user could actually carry out the function, training and documentation could reach a level of efficiency that might be comparable with live instruction by teachers and experts.

2.4 ERROR messages

Error messages are one of the most discouraging and problem filled areas of documentation. The error messages I have encountered are both discourteous and uninformative. Messages like "OPERATION ABORTED" leave me with the feeling that I may have just participated in a sleazy and illegal procedure. Of course, these messages are intended for experts and other cognoscenti, but on-line documentation may mean that novices will encounter these obscure comments more often. This seems to a fairly good reason for integrating these Error messages with the larger body of HELP functions that are available. The error message might recommend which HELP function to use. Even if documentation is not on-line, it would seem to be a very useful thing to have error messages refer to page numbers in a documentation manual that could provide some timely assistance.

2.5 Personalizing Assistance

Much documentation is written in a very impersonal style. There are some indications that this style is inappropriate when documentation is on-line. Users have a very strong tendency to anthropomorphize their interactions with computers: they tend to think of computers as another human being. Computer programs are beginning to take this forceful tendency into account by personalizing interactions. Computers will often ask for a user's name and respond with that name in providing information. It is my judgment as well as others' (c.f. Nickerson, 1981) that this superficial personalization may carry some heavy costs as well as benefits. For instance, it encourages users to think that computers are smarter than they really are. But in terms of on-line documentation, this kind of personalization is probably necessary.

Providing help and assistance is something that always needs to be done gracefully, especially when it comes from an inanimate object. Terse error messages are no way to soothe someone's frustration at being unable to complete a task smoothly. So personalization should be carried more deeply into sensitive social interaction. Liberal use of thank you, sorry, and please is probably highly appropriate. It would probably be wise to adapt the documentation style to the user's style; on such characteristics as politeness, longwindedness, and directness. But this is taking us a little beyond the scope of this paper.

2.6 Natural Language

Artificial intelligence and psycholinguistic research has made significant strides toward developing methods for natural language understanding in computers. It is not unreasonable that this research might soon find applications in on-line documentation. This is particularly important for novice users of the systems. It is probably most important for structuring HELP functions. There are also functional alternatives to the implementation of complex systems for natural language understanding. One alternative is to compile a complete list of the questions that novices tend to ask about a particular system. This approach has been implemented by Ford (1982) at Johns Hopkins with checking account and library indexing programs. The limitation to this approach lies in the memory capacity of the computer. This is a significant limitation, but complete language understanding is not the objective of this approach. A limited understanding can still be very useful for novices.

3. SPECIFIC REQUIREMENTS

3.1 Overview

Documentation of an authoring system must take into account situations that may arise in producing software for instruction. It is not clear that these are general issues, although some aspects may be applicable to the documentation of a large variety of programs.

3.2 Defining the User

In order to individualize on-line documentation and make it optimally functional, it is important to define the purposes the software is to serve for a particular user. In an authoring system, the documentation should first assess whether the user is a teacher, student, or courseware writer. Not only should specific software functions be tied to this assessment (so that a student cannot change the answers, for instance) but the documentation should be keyed to this assessment too. If a metaphor is being used to make the software functions more comprehensible, then the on-line documentation should be phrased in that metaphor.

In an authoring system there will be many kinds of teachers and students using the system. To some extent their characteristics cannot be predicted. But there will also be a large number of different courseware writers, and these can be more easily predicted and described. For instance, there will be instructional experts, programmers, media specialists, cognitive scientists, and graphic designers. The documentation should support the demanding characteristics of each professional.

Defining the user also means defining the expertise of the user. A simple menu may be the most efficient information that an expert can have, but a first-time user may well need very lengthy explanations and examples to demonstrate specific functions. Defining the user can therefore be a very complicated exercise, but a very important one. There is very little in this world that is as frustrating as asking for a HELP function that gives messages that are not very helpful. Of course it can be unhelpful in many ways: for instance, it can provide information I already know, or it can provide information that I do not understand. Both are frustrating. The best kind of documentation would provide information that I need before I know enough to ask for it. It seems to me that this should be possible, if the needs and knowledge of the user are carefully defined beforehand.

3.3 Defining the Structure

It is not enough that documentation should help someone use the system, it should also make the user expert. In part, this can be done by defining the user, and if he is a novice, providing instruction that makes him more expert by training and education. However, the system documentation should also help structure the system product so that the final result has expert characteristics. It is difficult at this point to distinguish between system function and documentation, but this will be an ongoing difficulty of distinction in any on-line documentation system.

Since documentation is integrated with system training and performance; the more thorough the integration the more difficult it will be to draw a line between system function and documentation. Since an authoring system is designed to produce more software, the documentation should provide a natural encouragement of good design practices like structured programming, and object-oriented code. Portions of courseware will be reusable in different implementations, and modular construction will make the transfer of frequently used portions more convenient. Similarly, documentation will need to provide advice about how to make the software more tansportable, so that it can be implemented on a wide range of currently available machines. These are only a few of the structural characteristics that documentation can help achieve.

3.4 Defining the Feedback

An authoring system that responds to the needs of a student actually using the educational software will require feedback and analysis systems that go far beyond the traditional requirements of HELP functions. A student will always need access to a real teacher, who can answer a question like "What did I do wrong?". However, some level of machine tutor will also need to be built into the system. Intelligent coaching can be added in many ways. At a primitive level, a trace of the sequence a student followed can provide some meaningful feedback. At a more complicated level, the system can analyze the errors a student made to try to find consistent errors or "bugs" in thinking (Brown & Burton, 1978; Feurzeig, Horwitz, & Nickerson, 1981). On-line documentation at this level will require very sophisticated understanding of a student's needs in order to explain how to use an intelligent tutor like this. Clearly, this kind of documentation will grow in complexity with the increasing level of understanding that a student has. The documentation itself will become part of the individualized instruction. One of the most important aspects of this documentation may well be the growth in self-knowledge that it promotes.

3.5 Defining the Input/Output

One of the most difficult and important aspects of effective instruction lies in the appropriate selection of input and output devices appropriate to a particular activity. With the large number of devices currently available (keyboard, joystick, lightpen, digitizer, mouse, and voice input; printer, videodisc, plasma, graphic, music and voice output) documentation of these possibilities has become particularly complex, especially if the documentation is designed to make the process more efficient and effective by providing timely advice. The complexity of this documentation can be suggested by examining just one of these areas: graphic display. Just imagine the difficulty of introducing novice users and documenting the following functions:

- o Multiple type fonts available at full speed for screen display;
- o Two-dimensional objects made up of points, line segments, circular arcs, and spline curves are able to be translated, rotated, dilated, contracted, and stretched anisotropically;
- o Three dimensional graphics can be used to isolate components, compute areas and volumes, and demonstrate multidimensional functions;
- o Animation can occur in real time;
- o Complicated, high resolution displays can be retrieved on-line from videodiscs and ROM packs or bubble memory modules;
- o Graphics packages for bar and pie charts and graphs are available with 3-D display capabilities.

The enormous capabilities that computers bring with them to almost every function of instruction, like these display capabilities, place a heavy burden on documentation. It is a burden that may well be possible to carry only if the documentation is on-line and integrated into the system's function.

4. CONCLUSION

4.1 This paper has presented a brief overview and suggestions for incorporating human factors considerations into on-line documentation. These suggestions include a guiding metaphor for each user; integrating HELP functions into the error messages for training purposes; personalizing interactions and making them conform to the rules of good conversation; using natural language protocols, particularly with novices; and providing useful, system-initiated feedback and information that reduces the enormous complexity of the tasks. These suggestions were discussed within the context of an educational software authoring system, but the discussion was intended to be general and apply to many computer systems.

An additional and very important point that needs to be raised in conclusion, especially in the hope of stimulating further discussion, is that this paper has not begun to address the complicated problems of potential relations between on-line documentation and print documentation. It should be clear that this has not been an argument for replacing all hard-copy with on-line documentation. There is probably a need for both. On-line documentation is designed for users at work on the computer, but clearly students and others will do work at home or in other ways removed from the computer, where hard-copy and manuals are clearly cost-effective. Just because information can be presented on-line does not necessarily mean that it should be. The particular uses that each serves best still need to be thought out carefully, and need to evolve in response to the changing circumstances of computer use.

REFERENCES

Brown, J.S., & Burton, R.R. Diagnostic models for procedural bugs in basic mathematical skills. Cognitive Science, 1978, 2, 155-192.

Feurzeig, W., Horwitz, P., & Nickerson, R.S. Microcomputers in education. B.B.N. report No. 4798, prepared for the National Institute of Education, 1981.

Ford, R. Human engineering problems of the person-computer interface. A paper presented at a meeting of the Software Psychology Society, 1982.

Heines, J. M. The personal computer as an appliance. Problem I: Integrating training and documentation. A paper presented at the International Conference on Cybernetics and Society, 1981.

Nickerson, R. S. Why interactive computer systems are sometimes not used by people who might benefit from them. International Journal of Man-Machine Studies. In Press, 1982.

NBS FIPS Software Documentation Workshop, March 3, 1982

Improving the Quality of User-Level Documentation
Remarks on Four Workshop Papers

Michael B. Feldman

Department of Electrical Engineering and Computer Science
School of Engineering and Applied Science
The George Washington University
Washington, DC 20052

The four papers in this session are an interesting group. Further, the session itself is interesting in that it is devoted to a discussion of user documentation in a workshop whose primary purpose has been to consider problems in developer or maintainer documentation.

Types of Users and User Documentation.

Let me first distinguish some types of documentation by making an analogy with the automobile. I know of at least five different major kinds of documentation associated with cars:

- Design drawings, made up by the designer but not detailed enough to build the car from;
- Shop drawings or blueprints, from which the (subsystems of) the cars are actually built;
- Mechanic's guides, or "shop manuals" which enable a professional mechanic to repair or maintain the vehicle;
- Repair books intended for the amateur mechanic, which do not detail repairs too delicate for the do-it-yourselfer to attempt;
- Driver's manual, written in lay terms, which tells how to operate but not repair the automobile.

In general, this workshop has been concerned with documentation analogous to the first four types, intended for people designing, building, maintaining, and repairing programs. The present session is -- happily -- concerned chiefly with documentation analogous to that in the last category: documentation for people using programs.

Now let us limit the scope of the discussion a bit by considering only users of interactive programs or systems (this is not very limiting since most programs are interactive these days anyway). In the research we have been doing, we have found it convenient to divide these users into five groups:

- Computer professionals like programmers or console operators;

- o Trained operators in routine environments, like bank tellers, airline reservationists, data-entry clerks or word-processor operators;
- o Trained operators in high-risk or complex environments, like power-plant operators, air traffic controllers or military command/control operators;
- o Casual users, like attorneys or congressional staff using on-line retrieval systems, professors writing papers on word-processors, or lay persons using automated teller machines;
- o Data analysts like statisticians or financial analysts: the so-called "powerful minds using powerful tools".

With the possible exception of the third category, a typical interactive system may have users in all of the classes, and at different levels of expertise. It is thus clear that we face a non-trivial problem in organizing documentation to suit the special needs of each class.

Can Programmers Understand User Documentation?

Experience also shows that technically-trained computer specialists do not intuitively know how to prepare documentation for any of these groups. It is a common saying that programmers don't like to document -- even for each other -- but what is left out is that we often don't give them training in how to document, or even in understanding the class of users for whom they're documenting.

I am reminded here of a group of experienced programmers responsible for a paycheck-writing system in a data processing service bureau I once worked for. When asked what their product was, they responded "programs", and were quite taken aback when we assured them that their product was "paychecks", and that the client was not concerned whether those checks came from a computer or a quill pen, only that they were delivered on time with the right numbers on them! This bit of consciousness-raising did help to put them on the track to understanding who their users and their customers were.

In the university, while we do not usually have "real" users for whom the students are producing programs, we do have some very real carrots and sticks, called grades, that we can apply to make certain that our teaching about documentation is taken to heart. Indeed, I and many of my colleagues assign as much as 60% of a project grade to the quality of maintainer and user documents. When exposed to good examples of documentation, carefully taught how to prepare them, and given immediate positive and negative feedback about the results, even programmers can learn how to do it.

Comments on the Session Papers.

Douglas Hines has given an interesting presentation on user-document preparation in the House of Representatives. His experience shows that involving the technical person to write the first draft of a document, then a representative of the user to refine it, pays off handsomely.

Further, he makes a good point in noting that documentation has often not been used in training new users; it is clear that incorporating the user documents into user training helps the users to learn their way around the documents at the same time they're learning their way around the system. This is important.

I had hoped to hear more discussion about on-line documents in the House. Their retrieval systems appear to be really modern and interactive from the start; I'm thus a bit surprised that there is so much emphasis on paper documentation.

Modern research on interactive system development suggests that the right "nooks" to on-line documentation need to be integrated at design time; I hope to see more of this notion applied in the government information systems of the future.

It is not in the least to denigrate the fine work done by Hines and his colleagues to note that papers such as this were being written ten and fifteen years ago. Quite the contrary, it is clear that the need for papers like this is still great; users are not yet satisfied with the quality of their documentation. Hines' work is then evidence of the persistence of a knotty problem.

I have attended a number of talks by Aaron Marcus, for audiences varying from a small research group (my own) to several huge conferences. Every time I see his visuals, hear him talk, or read his papers, I am intellectually challenged anew. As a dyed-in-the-wool computer specialist, I am always amazed at how little I really know of the world of graphic presentation (indeed, how little we in computing know collectively about that world).

Marcus' work in interpreting graphic arts to computer professionals -- and computers to graphic artists -- is very important in this era of relatively inexpensive but powerful graphic displays, and in an era of increasing concern about the human factors of user interfaces and the usability of user documentation. His position as a working member of a group building interactive systems is a good example of the steps enlightened technical managers can take to improve the visual quality of their systems, and thus their acceptability to the user communities for which they are intended.

The university research group I direct with my colleague James Foley -- a world-class computer graphics expert -- is working on tools to improve the quality of user interfaces, including on-line documentation. The group has been exposed several times to Marcus' teaching, and their very adept incorporation of many of his recommendations into their own system designs is good evidence that hard-core programmers can easily learn to apply good graphic design principles for use even on simple alpnanumeric terminals or teleprinters. I hope to see much more of this "technology transfer" between computer people and graphic-arts people in the future.

Charles Dickson has given us another example of the benefits of cooperation between computer specialists and people with other talents -- here cartoonists -- to produce big improvements in the quality of documentation.

From the comic-book manuals designed for semi-literate World War II draftees, Dickson has taken us to IBM's experiment with comic-book manuals for some of their researchers who are very literate indeed. If a comic book style can make learning to use a new piece of equipment fun, or will encourage users to read their documentation, then by all means let us experiment more with it. The success of The Fortran Coloring Book by Roger Kaufman in introducing college students to the ins and outs of this widely-used programming language serves to emphasize the potential that is there.

Dickson's comment that documentation standards ought to be more encouraging to those wishing to experiment with less conventional styles deserves careful consideration by the committees which draft these standards.

Finally, Joseph Psotka has given us a provocative glimpse into the world of the behavioral scientists and their thoughts on how to incorporate sound human factors into on-line documentation.

As we move increasingly into a world of interactive computer systems used by people who are not computer experts, on-line documentation will become a key issue. Indeed, it has become clear from recent research that the separation between on-line documentation and the on-line system itself is unwarranted. More and more, on-line documentation is an integral part of the system itself, and needs to be carefully considered from the very earliest stages of design of the system.

Psotka's comments about the unhelpfulness of "HELP" are especially important; so are his remarks about current systems' forcing the user to abandon the context of his problem to request HELP. Every system designer should take a long look at the recent developments in the field of so-called "personal work stations" like the Xerox Star, to get ideas about how on-line assistance can be incorporated into the dialogue through windowing and context retention.

For detailed discussion of this and related problems, the reader is directed to the recent book Software Psychology by Ben Shneiderman, and to the classic work The Psychology of Computer Programming by Gerald Weinberg.

Summary.

All in all this is a fascinating and provocative group of papers. Our attention has been very effectively focussed on several of the many important issues in user documentation. There are a lot of suggestions here to take to heart.

Session H: Quality Assurance for Documentation

Introduction

Virginia C. Walker

Quality Assurance Division
Office of Statistical Standards
Energy Information Administration
Department of Energy

Documentation and its quality is often predictive of the quality of the system that it purports to document. This session approached quality assurance for documentation from several directions, remembering that to perform quality assurance measurement, one must have in mind quantitative and qualitative criteria.

In times past, there has been general concensus that people could agree what a document ought to contain, via the checklist or table of contents approach, but how to universally decide whether a document was good or bad remains elusive. Documentation could be ascertained to be good, or bad, on a case by case basis, but the reasons why it was good or bad were equally hard to describe.

In this session, the measurement of documentation quality was approached from several directions. Richard J. Thompson described an effort to evaluate the documentation at the Chemical Abstract Service; Caroline S. Levenson of Edition, Inc. described her firm's experience in preparing user documentation for a varied clientele and relevant aspects common to them; and Herbert Hecht of SoHaR, Inc. proposed a management tool for successful achievement of requirements documentation. Then John R. Gabriel of Argonne National Laboratory provided some general observations on documentation standards.

Participants in this session agree that the production of good documentation is not an incidental byproduct of software development; concentrated effort must be given to achieve a useable result, one which takes into account human factors and the realities of continued change.

NBS FIPS Software Documentation Workshop.
March 3, 1982

Auditing Systems Documentation

Richard J. Thompson

Manager, System Records
Chemical Abstracts Service

Chemical Abstracts Service was an early proponent of and has a continuing commitment to standards and guidelines for data processing documentation. Recently, the organization performed an audit to assess the state of health of its system documentation and to determine its current effectiveness in a technical environment which did not exist when standards were first put into effect. The audit process, findings and analyses are presented as one company's set of experiences. It is hoped that these experiences will serve to stimulate others in examining their own systems documentation activities.

1. INTRODUCTION

Over a period of time, any system can become less efficient as it undergoes internal change, or takes incremental steps to track its external environment. At some point, a comprehensive audit of the current system and its environment needs to be performed to gain a new overall perspective to act as a benchmark for future improvements.

Chemical Abstracts Service (CAS) introduced extensive changes to its system documentation practices in 1972: A central library was formed to process and store computer system documents and to control changes to system libraries, content and delivery requirements were revised, and the Standards Committee was charged with the review of all new documentation requirements.

During the next eight years, CAS systems had progressed from a batch environment using a single, large-scale computer to a distributed, online data base orientation. The central library made similar technical advancements to modernize its internal processes in capturing the increased volume and varied types of system and program documentation. But in the transition, central library emphasis shifted to the operating level of documentation (versus general system level) needed for proper change control. At the same time, the role of data-oriented documentation became more active as the needs for greater file and database integrity materialized. Related database functions also matured which added new components to overall system documentation in addition to individual systems application.

Given these evolutionary changes, CAS management requested an audit to examine and assess the present state of EDP systems documentation. Implied in this overall objective was an assessment of the following characteristics of system documentation:

- o Identity and purpose of system documents.
- o Compliance of actual practice with CAS documentation standards.
- o Documentation quality in terms of coverage, accuracy, currency, timing, and utility.
- o Effectiveness of documentation with respect to cost and demands on the time of design and maintenance staff.

2. BACKGROUND

2.1 Scope of Documentation Activity

It's estimated that 25 staff equivalents are routinely involved in the generation and administration of CAS systems documentation. The central administrative function supports the activities of 150 analysts and programmers as well as operations and user staff. The documentation inventory involves approximately 200 systems, 12,000 modules, 8000 files and 3000 data elements. The administrative system was first established to be a storage point for system documents and as a focal point for program change control. It has since grown to provide a wide range of related functions which include library reference services, maintenance of program libraries, data base records, physical tape and disk management, file reorganization, recovery system design and operation, archive management and other recording and consultive services.

The current cost of systems documentation at CAS is almost equally divided between generation and administration of records. The cost of document administration has incrased marginally over the years with the addition of newly centralized functions. The cost of document generation is based on management's commitment to allocate 10% of all project development work for documentation tasks, and thus will vary directly with the number of staff involved in project activity.

2.2 Internal Documentation Guidelines

The CAS Data Processing Standards and Procedures Manual is the major reference for guiding the preparation and control of documentation. Documentation and other topics in the manual are regularly reviewed by a Standards Committee composed of technical and supervisory staff.

In addition to the Standards and Procedures Manual, lectures on standards and documentation are available on video tape and are mandatory sessions for systems development programmer trainees. The standards lecture provides a general orientation on the

reasons for standards at CAS, how to use the Standards Manual, how to locate forms and procedures, and an introduction to the standards committee and Central Library activities. The bulk of the documentation lectures emphasize techniques for making comments on listings.

All training programs are required to be documented and are regularly reviewed by Programming Groupleaders. Other documentation guidance and assistance for more experienced staff is on an informal peer basis.

Guidance is more adequately provided for detailed operating-level documentation (through training sessions, specific attention to forms, standards and procedures in the manual) than it is for general-level systems documents. The content of these general-level documents (user requirements, conceptual design, system summary, and user manuals) tends to be flexible and thus reflects the general guidelines and suggestions for topic content appearing in the manual.

3. THE AUDIT PROCESS

3.1 Audit Scope

The documentation examined in the audit is that associated with the development, operation, maintenance, and use of CAS computer systems. The itemized list of specific documents is lengthy, but in general it included systems and program documents, data base and run documents, and user and operating manuals. Collectively, these items constitute the CAS systems documentation end product. Control type documentation was addressed only to the extent that it contributed to the analysis of the documentation process. Thus, such items as project management/control documents, forms, logs, and transfer documents involved in the process of preparing the systems documentation end product were not treated extensively in the audit.

3.2 Audit Staff

The audit team included representatives from the systems development, operation, maintenance, and records areas of the organization.

Groupleaders from the Systems Development and Research Departments were also involved in audits of individual systems. Project Leaders reviewed the results of these audits and approximately 100 staff from 18 development, operating, and using departments participated in a survey of documentation usage.

3.3 Audit Method

Exploratory work was first conducted with R&D department and division management and an independent consultant to outline the purpose and scope of the documentation audit, and to develop and clarify sub-objectives. A detailed work plan was generated as a result of these meetings.

The CAS Data Processing Standards and Procedures Manual was reviewed to identify the items which constitute the systems documentation end product. This step developed a working control list of documents against which individual systems were audited.

Seven systems were audited for compliance against the control list. Auditors with various systems experience and perspective also reported on the quality, completeness, timeliness, and accuracy of the individual systems. Results were then discussed with the relevant project leaders. The systems selected included in-process and production systems, large and small, online and batch, and other varying characteristics to provide a reasonable sample for observing differences in documentation results.

Two hundred survey questionaires were issued to staff in 18 CAS departments to determine the predominant reasons for using documentation, the frequency of access of various documents, and the availability and utility of systems documentation in general.

Audit staff also collected other pertinent data to describe the scope of the documentation activity at CAS, its costs, and staff involvement; to generate critiques of various processes; and to collect suggestions for improvement.

4. AUDIT FINDINGS

Overall findings of the 1980 documentation audit indicate that the current CAS system of preparing and maintaining documentation still adequately serves the system development and operations process. The areas needing improvement focus on more prompt delivery of general level documentation produced during system development (vis-a-vis more specific operating documents). The survey also revealed how documents are most used by staff and, therefore, pointed the way in which improvements should be installed to gain the most benefit in terms of document utility.

4.1 Individual System Audits

Documentation audits were conducted on seven CAS systems. The systems characteristics varied as to development and production status, online and batch, 370-based and distributed, long and short development periods, and systems developed and documented by different R&D organizational units.

The audits revealed no documentation deficiencies that were peculiar to a given type of system. Each system had most of the specified documentation; but most systems were missing documents that the Standards Manual said should exist. (Note: "missing" is defined as not deposited in the Central Library).

The audits could not determine why any given system had relatively more comprehensive documentation, or more prompt delivery than another system. Highly visible projects with high development priority might be expected to have more complete and higher quality documentation. Yet, the chances seem to be even that full documentation will or will not be delivered. The only conclusive pattern is a lack of consistency in controlling the delivery of documentation.

4.2 Documentation Survey

4.2.1 Is Documentation Used?

The survey found that all the documents generated within the development, operating, and using range were being used. It was also found, not unexpectedly, that different groups of people used some documents more than others. Analysts and Managers mentioned the system description and program documentation most frequently; programmers overwhelmingly favored the module listing and Data Base Documentation; and users mentioned user manuals and the system description. In the composite group of 94 respondents, each of the 8 classses of documents received between 51 (run doc) and 70 (DB doc) mentions.

4.2.2 What is Documentation Used For?

One might expect that the original purpose of having documentation at all would correspond with staff's reasons for using it now, and this purpose would be to run a system or to make changes to it. It turns out that "changing the system" and "system orientation" are the leading reasons (a tie) why staff use documentation. "Operating the system" fell behind (by 60%) such choices as "designing new systems," and "debugging;" but ahead of "generating their own documentation."

Although the users need documentation to operate the system (60% of the respondents said so), they also need it for debugging, orientation, changing the system, and generating documentation. This interest indicates a larger scope of technical activity in the CAS user community than previously experienced due to the rotation and migration of data processing staff to user groups.

4.2.3. Which Documents Are Used Most Frequently?

The indisputable answer to this question is the module listing. This is based on responses to the question, "How often (expressed in percentages) do you access each of these documents?" Analysts and Managers use the System Description, and then the listing most often. Listings are by far the most accessed document for programming use. The user manuals and run documentation are the significant references for users. Weighted accesses

for the composite population show that the listing leads all documents by a wide margin; the remaining documents are in a narrow access range with user manuals on top.

The audit team noted that the utility of the listing has become more important since module documentation was recently incorporated into it via standard comments. This feature is also expected to improve the currency of that documentation.

4.2.4 Does Documentation Satisfy Needs?

Given a choice of "never/usually/always," the predominent opinion of staff is that documentation usually meets their needs. There were many qualified responses to this question ("usually, but ... "), and half the staff offered specific suggestions for adding to, deleting or expanding various documents. From these and other general comments, the audit team interpreted the response as an overall satisfaction with the existing documentation, but that staff feel there is a definite need for improvement.

Listings and data base documents received the highest marks in relevancy for staff needs. The system description, program and module documentation received the lowest. General comments reflected the opinion that there is little confidence in the currency or accuracy of this general level documentation.

At least half the time, staff go to some other person for help in answering their questions. This occurs in cases where documentation can't be found, or doesn't address a particular question. The source most often named was the "system expert," then the "user," then the code itself. Programmers preferred to ask other programmers or refer to the code, while the "users" main source of reference was a programmer.

4.2.5 What References Are Used in Preparing Documentation?

None of the respondents relied on any single source of help in generating their own documentation. Analysts and Managers and Programmers most often used the Standards Manual and existing system documentation for help; Users relied on other documentation and peers. In the composite sample, references were cited in the following order: other documentation, Standards Manual, peers, supervisor, no help needed (even these respondents checked a second source - usually the Standards Manual or other documentation), and training classes.

4.3 User Documentation

In conducting the audit, team members expected more vocal complaints about user documentation than were actually received from users. But, users sided with other staff in expressing major concern with the currency of other documentation as well as user doc.

Audit results were revealing in that users have the same functional reasons and access the full range of documents much as their analyst and programmer counterparts do.

This reflects a greater degree of staff experience and more project participation and which in turn indicates a larger scope of technical activity than expected in the CAS user community.

In addition to the several systems cited for which user documentation is non-existent, User's prime concerns were as follows:

- o More detail is needed, completeness of run documentation and flow charts.
- o The need for more user-oriented documentation and less computer jargon.
- o Better coordination of changes.
- o Identification of document authors and purpose of changes.
- o Better documentation of computer reports.
- o Removal of redundant or useless data.

5. ANALYSIS AND RECOMMENDATIONS

The recommendations of the audit team dealt directly with ensuring an improved level of currency of system documentation. It was recognized that the currency problem had many attributes. Documentation could be missing (from the Central Library), incomplete, lost, late and/or inconsistently revised.

In fact, one of the major symptoms of poor currency observed in the audits was that detailed operating and data base level documents were not consistently reflected in the more general systems and program documents. As a result, programs and data sets currently in use for a system were reflected in operating level documents but were not identified in the system. There were also instances where system A used copy code from system B and was unaware of subsequent changes to system B. Some detailed changes were observed which altered the logic of a program or function of a system, but documentation was not revised.

Specific recommendations were made to correct several administrative and control faults which contributed to the currency problem.

5.1 A Separate Documentation Manual

The CAS Data Processing Standards and Procedures Manual is a major reference source for guiding the preparation and control of documentation. Documentation gets extensive treatment in the Manual, but specific procedures are scattered and imbedded in other dominant topics. Given the level of indexing and cross-referencing in the manual, it becomes necessary to leaf through the Manual for a thorough and complete understanding of the subject. There are also many ambiguities in delivery requirements, document intent, audience, and responsibility for preparation.

A separate manual has been proposed for comprehensive treatment of the whole spectrum of documentation activity. This will sharpen the focus on the topic, become a single authoritative guide for training, and an ongoing reference.

5.2 Standardized Composition and Terminology

Elementary controls are being established which will eliminate problems caused by inconsistent use of terminology and document composition. For example, system documents delivered to a central function may use different titles. Further, they may be packaged in notebooks with still different labels. As a result, documents sometimes cannot be recognized for what they are. Positive statements will be used in following outlines, since it can't be determined if gaps in documents are a result of omission or commission. Other composition faults include the lack of dates or author names; missing tables of contents, bibliographies, and reference lists; shortcut methods (e.g. "see REFAID") which are not explained; and forward references (e.g. "see (other) Program X for halt log") which may not exist or may have been changed independently.

5.3 Unified System Document

A system's documentation becomes fragmented when there is no single storage point which collects all the information on a system and when existing documents are organized for ease in internal use or for individual department convenience. The Central Library itself will file module listings, program documentation, run documentation, data elements, run procedures, etc. in different areas. Pertinent system memos and reports may exist in other department files. Detailed run documentation, data set definitions, and file recovery procedures are processed and stored in still other organizational units.

This may not be a major cause for concern with knowledgeable staff available in-house who know what to expect in systems documents and where to go to find the missing pieces. However, it becomes a significant problem when it is necessary to transfer all written systems knowledge out of house to colleagues, archives, or users. With each occurrence, there is a non-trivial amount of staff time expended to: 1) identify what belongs to the system, 2) where to cut off the successive system/program dependencies, and 3) collect the information.

To correct this situation, CAS is reviewing its document delivery criteria for systems development and will emphasize the use of a single systems notebook which will include an inventory checklist of related documents, consistent use of bibliographies, standard nomenclature, and references.

5.4 Document Delivery Controls

General level documents generated during the system development process are typically delivered much later than system installation. Operating level documentation (data base and run documentation) is normally current because of its forms-driven processing, production involvement, and machine level control. The central Library may merely accept and

file documents that are delivered without any attempt to determine or enforce what should be delivered. Any of these factors prevent the Library from guaranteeing any level of integrity of system documentation.

CAS plans to continue to enforce documentation requirements for production software changes. Unless proper documentation is submitted, the job will not run: there is no appeal. Periodic audits for missing documentation will also continue. A standardized form of documentation checklist will be reviewed by Project Leaders and Managers for individual applications systems. The checklist will be used by the library to determine what documents should be delivered. A collection of checklists will form the basis for total inventory of CAS systems documents.

5.5 Document Trace Procedure

To some extent, the overall documentation environment will continue to be fragmented, and staff will still rely on their experience and knowledge of what data base reference aids exist or what information is contained in listings and program libraries. The audit team has proposed a written guideline which prescribes how to identify, trace, and collect the pertinent documents of a system. The trace procedure will describe common procedures of document administration and specify (e.g.) how to identify modules called by other modules, what Data Base or general purpose utilities are needed to run the system in question, where to find and how to use inventory lists and references which are located in other physical areas.

6. MANAGEMENT ENVIRONMENT

A final point worth mentioning is that no documentation function can ever expect even a near perfect document compliance and currency environment without strong management support. CAS is continuing its commitment to quality documentation by implementing the recommendations proposed in the audit and incorporating them with the results of a parallel effort to review and analyze the systems development methodology and its deliverable documents.

CAS expects that in implementing these improvements, it will be in a better position to support such new advances in its technical environment as Programmers Workbench, UNIX-based systems, office automation and electronic transfer of information.

Use of the Users Manual as a Quality Control Tool

Caroline S. Levenson

Edition, Inc.

The preparation of user documentation should be viewed as a quality control task. In this task, a documentation specialist reviews the developed system, its user interfaces, and operating procedures to verify that all functions are integrated properly and that user requirements have been met. However, to serve as a quality control vehicle, the Users Manual standards prescribed by FIPS PUB 38 must be modified to stress the user's information needs rather than the system's internal components.

Keywords: Quality control; Quality control tool; Users Manual; System Verification, User information.

1. INTRODUCTION

Computer documentation is usually the last task to be performed in a system development effort. While this is not ideal, it is a very real situation and can be used to advantage if the documentation process is used as a quality assurance or control task.

This paper discusses how the Users Manual in Federal Information Processing Standards (FIPS) Publication 38 (FIPS PUB 38) [1] could be used, with some modifications, as a guideline for this type of quality control. In this discussion, the current Users Manual contents are evaluated, and changes are suggested.

2. RECOMMENDATIONS

The following are recommendations for changes to the current guidelines for user documentation:

1. One or more Users Manuals should be prepared for any new system to describe the overall use and intent of the system and its products as well as all system interfaces and required procedures. Regardless of size, any system is a failure if its user(s) can not understand and effectively work with it and its data products.

2. The Users Manual should not be a restatement or copy of information presented in previous documents (e.g., Requirements and Specifications) in the system life cycle.

3. A documentation specialist(s) who has not been responsible for hands-on development work should be assigned the task of preparing the Users Manual(s). This person should be free of the assumptions and built-in partiality that exist for members of the development team and, thus, could review the system from a quality control perspective.

4. In developing user documentation, the documentation specialist performs a quality control function by verifying that the user's requirements were fulfilled by the developed system and that the procedures (e.g., data entry, report analysis, and operations) for user interaction are correct and can be carried out efficiently by the designated user representatives. This verification process should be thorough to be meaningful and may entail a wide variety of activities, such as (1) interviews with the user to review expectations and requirements and to determine the availability and skill levels of

persons who will interface with the system and (2) detailed walk-throughs of the procedures with existing test data or other specially prepared data.

3. THE QUALITY CONTROL FUNCTION

3.1 General

Documentation is currently considered a reporting task in which the development team passes along the information it has gained during the system life cycle. The drawback to this involvement in both development and documentation is that the development team is usually too familiar with the system and can not divorce itself from a development perspective. Also, a conflict of interest could arise if accepted system specifications are found to have incorrectly translated user requirements or were not updated with changes in those requirements. Consequently, the development team can not review what it has done with impartiality for documentation or quality control purposes. This is especially true with user documentation, which would require the development team to step out of its natural role and "think like a user."

Documentation can and should be more than a recording of the development team's work to date. It should be a final appraisal of the completed product and a description of how it fulfills all of the user's requirements, as well as a description of the internal system. It also should be prepared by a documentation specialist, who, like his counterpart in industry, is "outside" of the development team and checks the system to verify that expected capabilities, functions, and characteristics do indeed exist. This check can be as straightforward as comparing the developed system against Requirements Documents and System Specifications or as extensive as performing a series of walk-throughs of developed system procedures, interfaces, and products. In the latter of the two efforts, the documentation specialist is transformed into a mock user and simulates the "live" situations and problems with which the ultimate user and system must cope.

As discussed by the General Accounting Office (GAO) in a report to Congress on the problems of computer software development contracts [2], systems are often designed and developed hastily with inadequate or no testing and documentation. This leads to the unfortunate result that the users, who most often are the ones paying for the development effort, are left with inadequate systems or systems that they can not or do not know how to use. Modification of the Users Manual guidelines and the use of a documentation specialist to perform the quality control function should improve the quality and usability of developed systems and reduce the development team's post-implementation level of effort.

3.2 Documentation/Quality Control Tasks

The main activities of the documentation specialist in preparing user documentation and performing quality control should be to:

- Review existing documentation (e.g., Functional Requirements Document and System/Subsystem Specifications).
- Interview all user groups that will interface with the system plus selected development team members (as necessary).
- Review all system output for desired format, usability, and data quality.
- Check program code as necessary to obtain supporting information, such as calculations performed, validation criteria, error messages, and general processing characteristics.
- Prepare procedures for user interface points, including at least data preparation (completion of input forms), data entry, error correction, output distribution, and output analysis.

All of the above activities are necessary for good user documentation, because they look at system aspects that are visible to the user. In order to evaluate procedures that have been developed for the user, the documentation specialist must identify with the user, not the development team, and perform all involved steps. In so doing, he is the original integrator of the entire processing scheme and can decide whether or not all user requirements have been satisfied.

4. USE OF USERS MANUAL AS QUALITY CONTROL TOOL

4.1 Scope

The prescribed Users Manual in FIPS PUB 38 contains detailed information on the system software so "the user organization can determine its applicability and when and how to use it." This quotation from FIPS PUB 38 points out the main difficulty with the Users Manual standards both in terms of educating the user and in performing system quality control.

The goal of the Users Manual should be to show the user how the system is applied, including when and how, rather than providing him with the information from which the when and how can be derived. This information is gathered and verified by the documentation specialist from his detailed review of the system and all developed user procedures.

The Users Manual should be an outgrowth, not a repeat, of the Functional Requirements Document. However, parts of this document often are copied and transferred to the Users Manual without review or verification. Also, one Users Manual usually is not sufficient to cover all unique user groups and their information requirements and procedures in detail. A separate Users Manual should be prepared for each major interface point/user group or set of unique procedures. However, the FIPS PUB 38 Users Manual as it now stands is a general management overview instead of an individualized document for different user groups.

Each user group must understand its specific role in the processing scheme and must be given detailed instructions to follow or the entire system, though perhaps 100 percent code perfect and efficient, will not succeed. User ignorance, distrust, and misunderstanding are definite causes of many system failures and must be circumvented by improved user documentation.

The following paragraph describes the FIPS PUB 38 Users Manual and discusses what changes are necessary to better address user needs and serve as a guideline for quality control of the system. Tables 1 and 2 should be reviewed in conjunction with this discussion, because they show the current content outline for the Users Manual and compare the current guidelines with proposed changes.

4.2 Review of Users Manual Sections

4.2.1 General Information Section

The first section of the Users Manual is a synopsis of the system and its application by the user. Unfortunately, the prescribed contents do not emphasize the crucial difference between this and the general information sections in other FIPS PUB 38 documents and specifications.

Because of this similarity, this section is often copied from other existing development documents, even though the intent of this manual should be user- rather than development-oriented. This section should tell the user why the system exists and why he needs it as well as how the system will help him to do his job better. This is far different from simply summarizing the "application and general functions of the system" as is prescribed in FIPS PUB 38.

TABLE 1

FIPS PUB 38 Users Manual Contents

SECTION 1. GENERAL INFORMATION

1.1. Summary
1.2. Environment
1.3. References

SECTION 2. APPLICATION

2.1. Description
2.2. Operation
2.3. Equipment
2.4. Structure
2.5. Performance
2.6. Data Base
2.7. Inputs, Processing, and Outputs

SECTION 3. PROCEDURES AND REQUIREMENTS

3.1. Initiation
3.2. Input
  3.2.1. Input Formats
  3.2.2. Sample Inputs
3.3. Output
  3.3.1. Output Formats
  3.3.2. Sample Outputs
3.4. Error and Recovery
3.5. File Query

TABLE 2

Comparison of Current and Proposed User Manual Guidelines

CURRENT GUIDELINES	PROPOSED GUIDELINES
1. GENERAL INFORMATION	
1.1. Summary	
Summarize application and general function of software.	Summarize information needs solved by system and describe user interfaces and uses of system products.
1.2. Environment	
Identify user organization and computer center where software is installed.	Identify user organizations and equipment used in system interaction; also summarize computer capability, location, and availability, as well as contacts for solving equipment and related problems.

TABLE 2

Comparison of Current and Proposed User Manual Guidelines (Continued)

CURRENT GUIDELINES	PROPOSED GUIDELINES
1.3. References	
List of project authorization, existing project documentation and reference documentation, and FIPS publications.	Same content as current guidelines.
2. APPLICATION	
2.1. Description	
Description of when and how software is used and unique support provided to user organization, including purpose of software, capabilities and operating improvements provided, and functions performed.	Same content as current guidelines, except for more emphasis on user information needs rather than software. Also, an Application section should be prepared to address each specific user group separately in different manuals.
2.2. Operation	
Description of operating relationships of functions described in Paragraph 2.1. Also, security and privacy considerations and flow charts or descriptions of input/output are provided here.	This area should be renamed "User Interface" to describe input/output interfaces rather than software functions and processing. As with all Application section paragraphs, it should address each user group individually.
2.3. Equipment	
Description of equipment on which software can be run.	Overview of the equipment used by the user in interfacing with the system as well as detailed instructions on use of this equipment.
2.4. Structure	
Show software structure and describe role of each component in operation of software.	If this area is to be included in the Manual, it should be put at the end of the manual, since it is system-oriented information.
2.5. Performance	
Describe performance capabilities of software, including quantity of input/output, response time, processing time, error rates, and qualitative information about flexibility and reliability.	Description of interface work cycle, starting with initiation of data through analysis and use of data products. Also schedules for work cycle stages. Ideally, this area should be a subset of 2.2, Operation. However, system-oriented information, such as that covered by current guidelines can be important and could be included in one system-oriented section at the end of the manual.
2.6. Data Base	
Description of data files being used by system and the purpose of each file.	Same as current guidelines, but including information on update, backup, and other functions related by the data base. This section should be at the end of the manual.

TABLE 2

Comparison of Current and Proposed User Manual Guidelines (Continued)

CURRENT GUIDELINES	PROPOSED GUIDELINES
	2.7. Inputs, Processing, and Outputs
Description of inputs, flow of data through processing cycle, and resultant outputs. Also relationships among inputs and outputs.	This area should be eliminated, since 2.2, Operation, should include this information.
	3. PROCEDURES AND REQUIREMENTS
	3.1. Initiation
Description of step-by-step procedures used to initiate processing.	Description of actual completion of any source documents, including both data content and coding and processing flow.
	3.2. Input
Definition of requirements for preparing input data and parameters.	Since 3.1, Initiation, covers the preparation of the input data, this area should cover the actual procedures for data entry. These procedures must be step-by-step details of what the user must do to enter data and what the system will respond with, if applicable. If error correction is not elaborate, it should be included in this area instead of 3.4, Error and Recovery.
	3.3. Output
Definition of requirements relevant to each output.	Same content as current guidelines, but with emphasis on analysis of data content and use of data.
	3.4. Error and Recovery
List of error codes or conditions generated by software and corrective action taken by user.	This area should follow 3.2, Input, immediately or, in the case of basic correction of input data, should be included in 3.2. The content is similar to the current guidelines, but emphasizes procedures for error correction and recovery before listing error codes and detailed corrective actions.
	3.5 File Query
Detailed instructions for initiating, preparing, and processing query of data base wherever applicable. Also instructions for use of terminal if necessary.	Same content as current guidelines, but instructions for use of terminal should have been described in the Input area. Thus, the content should reference basic instructions in that area while stressing only query-related instructions here.

The following bulleted items describe suggested changes in terms of the major questions that a user wants answered in this section.

- Why. This should be answered by 1.1, Summary. It should tell why the system is being developed, which, generally, is to help the user to do his job faster and more efficiently or productively.

- How. This should be a description of the major functions and capabilities of the system plus a reiteration of the authorizing publications or documents in 1.3, References. However, to eliminate the need for updating due to changes in the documents covered in 1.3, references to, rather than summaries of, such documents would be preferable.

- When and Where. These questions should be answered by 1.2, Environment, which should be more than an identification of user groups and the computer center. It should provide the timeframe (e.g., quarterly, monthly) of a processing cycle and pinpoint the individual user groups involved in the function or interaction being documented.

  This section should also describe the major user interface points in general terms and describe how these interfaces are achieved (e.g., online terminals, batch processing, and remote printout and delivery).

4.2.2 Application

This section presents an overview of how the system is used. Currently, it combines all user groups into one composite user and tries to describe the duties and processing schedules of this fictitious user.

The Application section should be divided into separate sections with appropriate information for each user group mentioned in the General Information Section (as revised). Or, if user groups are distinct enough and the interaction procedures warrant it, each user group should be addressed in a separate manual entirely. If the FIPS PUB 38 standards were changed to prescribe one Users Manual for each distinct group, such as the day shift of a keypunch division or the budget analysts within an accounting department, this section would address each subgroup within the major user group.

Other recommended changes for this section are described in the following bulleted items.

- Section 2.2, Operation. This subsection of the Users Manual should be renamed "User Interaction" (or "User Interface"), because it should describe the input and output interfaces rather than the operation of the software. The user, after all, is most concerned with what he must give the system and what he gets back from it (and other users). He is secondarily interested in how the system functions in receiving and producing data.

  A graphic illustration of the user processes providing the input and the system and related processes producing the output would be very useful here. In addition, expected quantities, timeframe, and a description of the normal processing cycle are very important to the user and should be included in this section.

- Section 2.3, Equipment. The equipment used by the user may include online terminals or require the establishment of procedures for interaction with the equipment. These should be outlined here and detailed in Section 3, Procedures and Requirements. For example, users may never have used online terminals before and the Users Manual must tell them how to do so as well as how to work with the application software. It also must tell the user about equipment-related considerations, such as who to call if equipment fails or if the application software is "down." Indeed, it should help the user become

knowledgeable enough to resolve minor equipment problems himself and to at least try to determine whether a problem is software or hardware caused. This is very important to making the user independent of the development team and confident that he knows something about his system and equipment.

- Section 2.4, Structure. This section should be placed adjacent to Sections 2.6 and 2.7, since they all relate to system functions and requirements. (However, if proposed changes to 2.7 are accepted, that section will no longer exist.)

4.2.3 Procedures and Requirements

This section presents information about "initation procedures and preparation of data and parameter inputs for the software." This is reasonable, but it does not take the user far enough. The user needs to know how to initiate processing (including creating source documents and entering data into the system), but also needs to know how to analyze any system feedback (such as errors and general system-related information) and how to understand his output. Although these items are covered in the body of the section, their lack in the overview points out again the need for making the Users Manual more user-oriented.

Specific changes suggested for this section are described in the following bulleted items:

- Section 3.1, Initiation. The majority of this section should follow the discussion of system input (Section 3.2). However, an overview of the general processing flow, such as a flow chart or other visual depiction, plus a brief narrative should be left here.

- Section 3.3, Output. The main change to this section is the need to emphasize how the output data is used. For instance, Section 3.3.2.c describes some characteristics of the sample report, but it does not describe any interrelationships of reports or the use of calculations or formulae in producing the data values. Also, the definition of the sample report does not stress the overall meaning and use of the output, which is very important when many types of outputs are being produced for the same user. Also, output formats (3.3.1) without meaningful data are useless. The definition of output contents and format should accompany the sample reports.

- New Sections. An additional section is needed to describe computer-related capabilities or requirements associated with the input and output of the system or the user's interaction with the system. Examples are the procedures for obtaining general system information (e.g., system messages and processing queue information), maintaining a logging system for input, and distribution of output to the user.

  Also, appendices of error messages and correction procedures and data code tables should be included in the Users Manual. A glossary, wherever possible, would also be helpful to the user.

  NOTE: If the appendix approach to providing error messages and correction procedures is used, it should be for situations where the number of error codes is voluminous and would be too bulky to include in the middle of the manual in Section 3.4.

5. SUMMARY

Although the current FIPS PUB 38 guidelines prescribe a Users Manual with a great deal of information, they do not provide it in a manner that is appropriate for the system user. Indeed, much of the information itself is about the workings of the system rather

than how the user can work with the system. Therefore, the current guidelines need to be rewritten to cater to the user's information needs. By improving the quality of information that is supplied to the end user, a revamped set of guidelines should increase the usability of systems and thereby ensure that the user gets what he pays for.

The preparation of the Users Manual is a task best suited to an individual who is not responsible for developing any part of the system to be documented. This individual is the documentation specialist, who must identify with the user and be able to document all procedures, however small, that must be performed by the user in interacting with the system. In documenting these procedures, the documentation specialist must simulate the user environment and thoroughly check each procedure to ensure that it as well as the system has been developed properly and answers the user's stated needs.

The assignment of the documentation specialist to the task of preparing the Users Manual enables the documentation process to become more than a recording of facts and procedures. It enables the process to serve as a quality control on the system, user interfaces, and procedures. This quality control effort has not been adequately addressed by the data processing community. However, the marriage of the user documentation and quality control efforts into a single task makes this function more feasible, while increasing the accuracy of the documentation itself.

6. REFERENCES

1. National Bureau of Standards, U.S. Department of Commerce, Guidelines for Documentation of Computer Programs and Automated Data Systems, FIPS PUB 38, U.S. Department of Commerce, Washington, D.C., February 15, 1976, 55 p.

2. U.S. General Accounting Office, Report to Congress: Contracting for Computer Software Development -- Serious Problems Require Management Attention to Avoid Wasting Additional Millions, U.S. General Accounting Office, Washington, D.C., November 9, 1979, 84 p.

REQUIREMENTS DOCUMENTATION - A MANAGEMENT ORIENTED APPROACH

H. HECHT

SoHaR Incorporated
Los Angeles, California

ABSTRACT

The needs of post-design software life cycle phases are not met by present requirements documentation, and it is suspected that large economic losses are thereby being incurred. Requirements documentation is not being kept current, it is frequently too detailed, and the format inhibits use by the management levels that most need it in the later phases. To overcome these obstacles, a hierarchical structure for software requirements documentation is proposed that (a) limits the size of each volume so that it can be easily handled and read, (b) addresses specific information needs at each management level, and (c) is easily maintained. The hierarchical documentation is supplemented by a single volume that contains general project information. Both the structure and the content of suitable documentation are described.

Key Words: Requirements documentation, software management, software maintenance, software requirements.

1. INTRODUCTION

In the course of a survey of software tool usage conducted for the National Bureau of Standards, software developers were asked the question "What are the major software problems in your environment?" Of 17 individual problems identified in response to this question, the largest number, seven related to requirements, and the next highest category was maintenance for which four problems were mentioned [HECH81]. The participants involved in this survey were too few to permit very conclusive statements to be formulated. However, it must be surmised that requirements and maintenance together (and these are related, as will be shown) are indeed high on the list of problems that beset the software community.

The detailed complaints voiced in the requirements area were that requirements are not stated in a form suitable for software development (this will hardly come as a surprise), that they are not kept updated after the initial design phase, that even had they been current, the documents are not in a format useful for maintenance organizations, and that the most pressing questions that arise in later life cycle stages are not addressed. The maintenance problems mostly related to the fact that the available documentation was too detailed to provide the needed overall perspective for personnel who have not participated in the original development -- read: lack of requirements documentation.

The larger manifestations of the problems voiced in the survey are familiar to most software managers responsible for large systems:

* Inability to improve the performance of a frequently used module because no one is certain why obviously inefficient control or data structures, multiple initializations of a single variable, and multiple loops for what appears to be a single iteration sequence are necessary

* the necessity to guess the amount of effort required to achieve a new interface or to implement a new function because the time available for furnishing an estimate does not permit digging into the detail level (and that is the only one for which current information exists)

* the large effort required to develop requirements for a new system that is intended to be functionally equivalent to an existing one.

The root cause of these problems is that most requirements documentation (or what survives of it after the initial design phase) is being prepared by programmers for use by programmers. These documents do not address the needs of managers who have to make decisions about program redesign, program enhancements, or possible replacement of an entire software system. Yet, these decisions have very large economic consequences, and the lack of information is very costly. On the other hand, furnishing this information, if it is made an integral part of the software management process, is quite inexpensive.

The thesis of this paper is therefore to suggest an economical means for implementing management oriented requirements documentation that

* supports the needs of the post-design phases
* is easily maintained
* serves as top level of an overall requirements structure.

The purpose to be addressed by the proposed documentation is amplified in the following section. A specific structure and formats for the new documents are then described, and suitable interfaces with existing formats for requirements are discussed. It is believed that the resulting documentation will be especially suited for medium and large scale software or total information processing systems.

2. USES OF REQUIREMENTS DOCUMENTS

The conventional view is that requirements documents are being generated during the Definition Stage of the Development Phase to guide later efforts in the preparation of specifications and other Design Stage documents. This constitutes indeed a major purpose of requirements documents, and it is the one addressed by significant studies in the field [YEH80, ALF077] and by software tools for the automation of requirements documentation [IEEE77]. But it is by no means the only one. Other important uses for requirements documents are:

Test Case Generation - initial and updates

Software Maintenance - identification of possible deletions

Major Modifications - definition of the required changes

Replacement - point of departure for new requirements.

All of these take place late in the Development Phase or after it is complete. It is frequently held that at that point in time there is no longer a need for requirements documentation since 'more definitive' (really meaning more detailed) reports and descriptions are available. Although the importance of requirements documents in later life cycle phases is acknowledged in several of the previously cited references and is the specific subject of a more recent study [DAV180], there has been little published on the format and content of requirements documents that addresses their use in the post-design period. Because of this neglect, requirements documentation is not kept current, and this can have very adverse cost, schedule, and performance impact on later activities as is shown by some examples below.

The generation of at least a portion of the test cases from the requirements document (as distinct from the specification or program description) facilitates the discovery of mistakes in the translation of requirements into the subsequent documents. Under most current practices the use of the requirements document as a basis for test case generation uncovers inadequate updating of the requirements more frequently than it uncovers design or

programming errors. Test cases are therefore generated without reference to the requirements, and an important source of software errors is being neglected in the test program.

Practically all software contains segments or features that support temporary requirements such as handling an old file format during the conversion period, servicing obsolete peripherals that are still used as back-up equipment, or generating data for a function that will shortly be transferred to another agency. If the requirements document identifies the temporary nature of these features, the code sections that implement them can be appropriately commented, and their deletion can be systematically accomplished as part of future software maintenance. Conversely, if such a designation of temporary requirements is not provided, and if later updates do not prune obsolete requirements, the code will soon become a jungle of unusued, sometimes even inaccessible, segments which at the very least cause inefficient use of memory, frequently entail performance penalties, and lead to maintenance problems.

Major modifications of programs may become necessary because of new legislation, a change in mission or weaponry in the military environment, or the introduction of new computer hardware. In all cases the availability of current and succinct requirements documents will facilitate (a) establishing the scope of the software modification, (b) assigning budgetary and management responsibilities for the execution, and (c) the integration of the software changes with correlated activities, such as hardware installation or a new pattern of report distribution. Use of existing requirements documents in the management of a major modification also provides good assurance that the documents will be kept updated as the modification is implemented. Where requirements have not been kept current, the modification must be planned on the basis of more detailed documents which are usually not suitable for use at the appropriate level of management. As a result, there is much greater chance that some necessary steps might be overlooked, that there might be inefficient assignment of responsibilities, and that functions no longer required will be carried over into the new version. Moreover, there are seldom adequate resources to generate comprehensive requirements documentation for the modified version, and as a result a further deterioration in visibility of requirements, and hence for rational management of the entire software effort, will have occurred.

The importance of current requirements documents for the existing system in the procurement of a replacement is often overlooked. Planning for the replacement of a major software system (frequently coupled with new hardware procurement) can occupy a significant fraction of the time of senior management. In Federal agencies this task is further complicated by the desire to avoid commitment to a single vendor at a very early stage. As a result, requirements planning is either carried out in-house (rarely possible because of the heavy drain on senior staff time), contracted out to a company not eligible to participate in the procurement (inevitably causing inefficiency in information transfer), or conducted in parallel by multiple vendors (with obvious cost penalties). Where the current requirements of the existing system are not well documented, they are at this point reconstructed in a process that by its nature is extremely inefficient.

Under current budgetary constraints it seems incredible that more attention is not being called to the many cost penalties that arise when requirements documentation is inadequate or not kept current. Yet, this condition is the rule rather than the exception. The explanation is most likely in the diffuse manner in which the lack of requirements documentation makes itself felt. Once the specification is written, there is indeed no activity that comes to a screeching halt because requirements are inadequate or obsolete. On the other hand, the effort required to generate good requirements documentation and to keep it current is easily identified and therefore frequenly avoided. It is believed that this effort is sometimes overestimated, and that current concepts of requirements documentation address an uncalled for level of detail. These statements are amplified in the followig section.

3. STRUCTURE OF REQUIREMENTS DOCUMENTS

Because the emphasis in this paper is on the use of requirements documents in the post-design period, when the need to access them will be sporadic, the following addresses

primarily the structure for hard copy versions. If the requirements documents can be kept resident on a computer, the automated search facilities that will then be available may make it possible to consider other formats. Again because of the contemplated sporadic use by personnel who will not necessarily be familiar with the specific software, or who may not be software professionals at all, the use of specialized requirements languages and of formal syntax will not be acceptable. Moreover, at the top requirements level (which is the crucial one in the present context) these procedures contribute comparatively little to clarification of a natural language text. On the other hand, the use of decision tables or of graphics may be helpful supplements to text statements.

Because of the concern with ease of use, it is desirable to keep individual requirements documents reasonably small, say between 25 and 50 pages of text. Obviously, this is not sufficient to record the requirements for even a medium size software system at the level of detail that will support the activities outlined in the previous section. The way out of this difficulty is to provide a hierarchy of requirements documents. Thus, the top level document will contain general requirements levied against the entire system (see Section 4) and those detail requirements which must be implemented at the system level, such as sequencing and abort criteria for the next lower level functions. For all other requirements, reference is made to the requirements document for the appropriate function. Two levels of requirements documents will be adequate for many applications; for exceptionally large or complex systems, three levels might be necessary. The hierarchical structure also improves the maintainability of the documents as is explained below.

Downward referencing should be by volume designation only, consistent with the general principle of information hiding in hierarchical structures. This also assures that detail revisions to lower level documents can be made without impact on the higher level document. On the other hand, to assure traceability of requirements, upward referencing should be very specific (to the lowest pertinent indentation in the parent document). Through use of a one-way cross-reference (parent paragraph to descendant paragraph), bound with the lower level document, it is then possible to identify all provisions in a given document that might be affected by a change in the parent. An example of a top level requirements structure and the use of forward and backward referencing is shown in figure 1.

It must be realized that the assignment of functions from a single higher level document to a number of lower level documents is a design activity. There is frequently considerable latitude in the partitioning, and hence opportunity for choices that will be questioned at a later time. A statement of the rationale for the assignment of functions to the individual lower level elements should therefore be included in each parent document.

The structure of each lower level document is also a design process, frequently tied to an acknowledged or unacknowledged perception of the implementation. It might be pedantic and unproductive to require a justification for each subdivision of a document that is not directly dictated by requirements at a higher level. However, where a document is structured after some deliberation, a brief notation of the rationale will be very helpful for future uses, particularly for those associated with major modifications or replacement.

The hierarchically structured requirements documents can be supplemented by a volume that contains information common to the entire project such as definitions, abbreviations, units of measurement, common file and report formats, and the identification of the common hardware environment. Provision of such a volume not only avoids repetition of these items in each requirements document (and in other software documentation as well) but also simplifies the updating when any of the common items are changed. A technical report currently in the draft stage by the American National Standards Committee on Information Systems (ANSC X3) can be used as a guide in the preparation of the common project information [ANSC80].

4. THE TOP LEVEL REQUIREMENTS DOCUMENT

Considerable effort has gone into defining what needs to be stated in a requirements document in order to enable the design of a program to render an accurate and efficient implementation of these requirements. The previously cited references, particularly

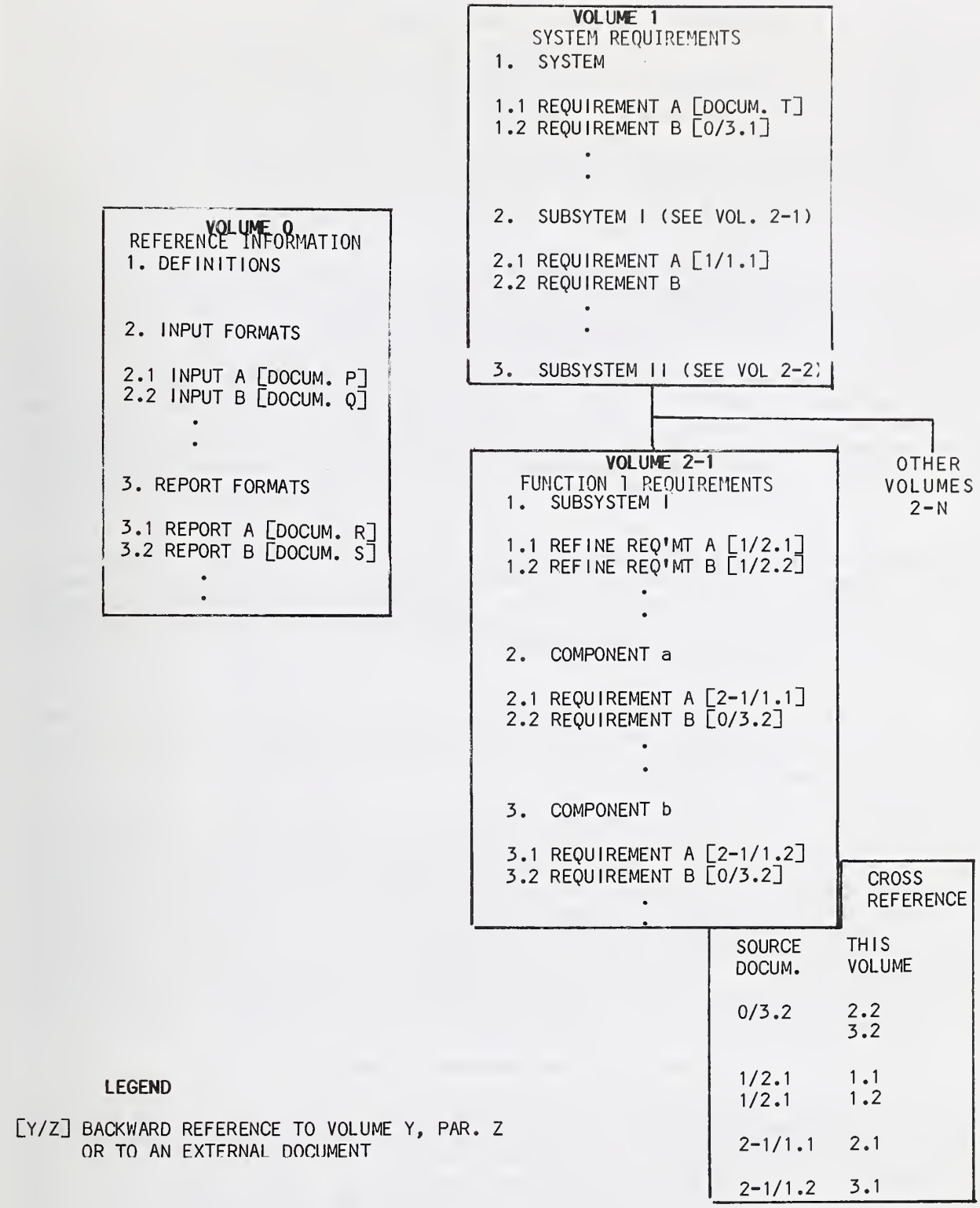


FIGURE 1 - EXAMPLE OF TOP LEVEL REQUIREMENTS STRUCTURE

[YEH80], provide good summaries of this work and the present contribution is not intended as an improvement in that area. However, even the best documents that are concerned solely with design do not provide much of the information needed for effective use of requirements documents in later life cycle phases. This information includes the authorization for the project, the specific legislative or administrative orders that it implements, and the constraints that were considered in the framing of the requirements. Many peculiarities of large programs arise from these items but their origin tends to be forgotten. Thus, when the causes or constraints are later removed, the program elements or structure that implemented them are allowed to persist, freqently through many revisions and even replacements.

This is one of the causes, possibly the principal one, of the loss of structure in large programs to which Belady and Lehman called attention [BELA76]. From their observations, the authors coined the 'Law of Increasing Entropy': "The entropy of a system (its unstructuredness) increases with time, unless specific work is executed to maintain or reduce it." If the unstructuredness is allowed to increase, the performance, the reliability, and the maintainability of the program can all be expected to decrease. On the other hand, the 'specific work' to clean up a program that has been in service for some time is likely to be a major effort. This effort can be reduced by current, complete, and accessible documentation. In the requirements area, much of the information needed to maintain programs well-structured (such as the items cited in the preceding paragraph), is readily available during the early stages of a project and can be recorded at negligible cost.

For the structure described in the previous section, the top level requirements document is the logical place for this information that usually affects the entire computer system (or at least the software). For military activities, a format that is suitable for capturing most of the pertinent data is the Navy's Operational Requirements Document for Tactical Digital Systems [NAVY74]. An example of a modification of this for use in a more general environment is shown in Table 1.

A number of the items in this table are also contained in the Feasibility Study Document format of FIPS PUB 64 [NBS79]. However, they are embedded there in a document that is intended to support the initiation phase and can not be expected to be kept updated throughout the software life cycle.

Two characteristics of the above outline deserve specific attention: most of the entries do not lend themselves to processing by current requirements analysis tools, and many of the areas covered are not usually of concern to programmers and analysts. Indeed, the top level requirements document is deliberately management oriented. It is intended to be prepared under the immediate direction of the management that provides the funding for the entire system development or procurement, and it is intended to be used by management responsible for the development of the individual major system segments, by the management responsible for maintenance, and then again by the management level that funds major modifications and eventual replacement (this will usually be the same organization that prepared it but most likely not the same personnel).

5. LOWER LEVEL REQUIREMENTS DOCUMENTS

The second level requirements document usually addresses software at the level at which the major development effort is segmented. The divisions can be by type of use (operational, operations support, maintenance, simulation and training), or by applications served. The top level document is fully visible at this stage, and all applicable provisions of the general requirements identified there should be referenced rather than copied. This eliminates the need for multiple documentation changes (possibly poorly synchronized or not carried out completely), when a top level requirement is changed. Allocations of requirements are normally identified in the top level document (e. g., allocation of overall performance requirements to the major segments), but the implementation of this allocation and the further allocation to lower segments need to be covered in the second level documents. Upward referencing must be complete and detailed (to the lowest pertinent indentation in the parent document).

As mentioned earlier, any refinement of a requirement, and particularly the allocation of

TABLE 1 - OUTLINE OF TOP LEVEL DOCUMENT

1.	INTRODUCTION
1.1	Purpose of proposed system and responsibility for implementation
1.2	Authorization for establishing requirements
1.3	Legislative requirements implemented
1.4	Administrative requirements implemented
1.5	Internal requirements of issuer of this document implemented
1.6	Scope of this document
1.7	Operational concepts (number and type of users, etc.)
1.8	Operating modes to be provided for
	a. normal operation
	b. operation during reduced equipment availability
	c. calibration or set-up
	d. training or simulation
	e. post-operational audits
1.9	Support Programs provided or required for
	a. program test
	b. equipment test
	c. personnel training
	d. performance monitoring, etc.
2.	APPLICABLE DOCUMENTS
2.1	Legislative, administrative, or internal authorizations
2.2	Standards, recommended practices or guidelines for technical aspects
2.3	Documents governing procurement and contracting
3.	OPERATIONAL REQUIREMENTS
3.1	General operational requirements (expansion of 1.7)
3.2.N	Operational requirements for each of the modes under 1.8
3.3.N	Performance characteristics for each mode
3.4.N	Accuracy and format requirements for each mode
3.5	Constraints (software & hardware cost, equipment size, portability)
3.6	Interfaces (may be separately stated for each mode)
3.7	Segmentation (time phasing for procurement or initial operation)
3.8	Other implementation requirements (including security and privacy)
3.9	Requirements for modification and expansion
4.	SOFTWARE REQUIREMENTS
4.1	General software requirements (language, file structures, etc.)
4.2.N	Specific software requirements for each mode under 1.8, including
	a. reference to next lower level document
	b. performance and other requirements allocated at top level
4.3	Software quality requirements
4.4	Overall test requirements (levels, independence of test organization
4.5	Overall documentation requirements
4.6	Reliability/availability/maintainability reporting requirements

requirements to lower level functions, really constitutes a design activity. As much as possible of the rationale for these design decisions should be documented, and the second level requirements document is a suitable repository of such decisions.

For military applications a suitable format for the second level requirements document is the Function Operational Specification in [NAVY74]. An example of an adaptation of this for use in a more general environment is shown in Table 2.

As might be expected, second level documents contain more of the information associated with the software development process. Many of the requirements listed in Table 2 (particularly under headings 3. and 4.) are equivalent to those of Sections 3 and 4 of the FIPS PUB 38 Functional Requirements Document [NBS76]. While these documents thus represent a transition from the primarily management oriented top level document to the technical software development requirements, their information content can be more readily supplied by project

TABLE 2 - OUTLINE OF SECOND LEVEL DOCUMENT

1. INTRODUCTION
1.1 Purpose of segment and responsibility for implementation
1.2 Functions served by this segment
1.3 Scope of this document
1.4 Operational concepts for this segment (number and type of users, etc.)
1.5 Operating modes for this segment
1.6 Programs (or major modules) contained in this segment
1.7 Support programs required for this segment

2. APPLICABLE DOCUMENTS
2.1 Parent document
2.2 Technical documents specifically applicable to this segment*
2.3 Procurement and contracting documents specifically applicable*

3. OPERATIONAL REQUIREMENTS
3.1 General (including allocation of functions to programs)
3.2 Basic operations (operator and user interactions with the system)
3.3.N Specific operations (for each program)
a. normal operation
b. operation during reduced equipment availability
c. calibration and set-up
d. training or simulation
e. post-operational audits
f. interaction with support programs
Under each of the above headings, describe input preparation, processing, use of output, performance and accuracy requirements, specific reliability/availability requirements, and identification of actions that must be avoided.

4. EQUIPMENT REQUIREMENTS
4.1 Minimum equipment requirements
4.2 Optional equipment for improved performance or further capabilities
4.3 Equipment requirements for expansion

5. INTERFACES
5.1 External interfaces*
5.2 Interfaces with other system segments
5.3 User interfaces (human engineering requirements)

6. OTHER REQUIREMENTS
6.1 Software quality requirements (including availability)*
6.2 Test and validation requirements
6.3 Documentation requirements
6.4 Reliability/availability/maintainability reporting requirements*
6.5 Security and privacy requirements
6.6 Requirements for modification and expansion*

7. CROSS REFERENCE (Top level provisions incorporated in this document)

* This information needs to be provided only if it differs from that in the top level requirements document

management than by software professionals, and it is therefore suggested that they also be prepared under the direction of a management function.

For very large software systems, a third level of requirements documents may be generated, primarily in order to keep the page count within a range that permits easy access to the desired information. Where the second level requirements documents have been broken down by applications area, the third level may be divided by type of use (operational, support, etc.), or vice versa.

The second (or third) level documents normally serve as inputs to specific requirements documents for each program to be developed or procured. Their purpose is to describe in detail the operational and functional requirements to a level of detail suitable for procurement, including criteria for the evaluation of each requirement. The Navy's Program Performance Specification [NAVY80] and the System/Subsystem Specification of FIPS PUB 38 may be utilized for this purpose.

Beyond their essential role for software development, the second level requirements documents should provide background information that is vital for the intelligent and efficient use of the software throughout its operational life. If the information is recorded as it is being developed (i. e., directives for planning and development, memoranda of understanding, etc.), the cost for the documentation is low. If it has to be retrieved at a later time, the cost inevitably rises. After some time the nature of the pertinent information becomes lost (no one remembers that the software structure was governed by a certain memorandum) and the information becomes completely unavailable. The effort for restoring structure to a program under these conditions usually exceeds the available resources, and a completely fresh start becomes necessary.

6. CONCLUSIONS

Requirements documents can provide valuable information for software lifecycle phases that follow design and implementation, but current approaches do not meet these needs and it is suspected this causes much economic loss. To overcome these difficulties, a hierarchical structure for software requirements documentation has been proposed that (a) limits the size of each volume so that it can be easily handled and read, and (b) addresses specific information needs at each management level. The hierarchical documentation is supplemented by a single volume that contains general project information, such as definitions, abbreviations, and report formats. This structure permits economical generation of the documents, provides traceability of requirements, and is easy to maintain because usually a change can be implemented by revision of a single document.

REFERENCES

ALF078 Mack W. Alford, "Software Requirements Engineering Methodology (SREM) at the Age of Two", TRW-SS-78-12, TRW Systems Engineering and Integration Division, Redondo Beach CA, March 1978

ANSC80 ANSC X3 "Draft Proposed X3 Technical Report, Guide for Technical Documentation of Computer Projects", Document X3K1 46a, April 1980

BELA76 L. A. Belady and M. M. Lehman, "A Model of Large Program Development", IBM Systems Journal, vol 15 no 3 pp 225-252, 1976

DAVI80 Alan M. Davis, "Automating the Requirements Phase: Benefits to Later Phases of the Software Life Cycle", Proc. COMPSAC'80, pp. 42-48, October 1980

HECH81 H. Hecht, "Final Report: A Survey of Software Tools Usage", NBS Special Publication 500-82, 1981

IEEE77 IEEE Transactions on Software Engineering, Special Collection on Requirements Analysis, vol SE-3 no 1 pp 1-84, January 1977

NAVY74 Department of the Navy, "Tactical Digital Systems Documentation Standards", SECNAVINST 3560.1, August 1974.

NAVY80 Department of the Navy, "Program Performance Specification, DI-E-2136A", MAT-09Y, June 1980.

NBS76 National Bureau of Standards, "Guidelines for Documentation of Computer Programs and Automated Data Systems", FIPS PUB 38, February 1976.

NBS79 National Bureau of Standards, "Guidelines for Documentation of Computer Programs and Automated Data Systems for the Initiation Phase", FIPS PUB 64, August 1979

YEH80 Raymond T. Yeh and Pamela Zave, "Specifying Software Requirements", Proc. IEEE, vol 68 no 9 pp 1077-1085, September 1980

Issues in Defining Standards for Documentation

*J. R. Gabriel**

Applied Mathematical Sciences Section
Applied Mathematics Division
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439

INTRODUCTION

Plato remarked that we are all metaphorically seated in a cave, chained to look only at the end wall, and seeing no more than the shadows cast by real objects in the sunlight from the entrance.

The three other papers in this session portray different shadows of the the same thing clearly and precisely. But the object has more than three dimensions, and I would like to discuss other projections I have seen over the years. A good standard takes account of all dimensions, and considers principal axes instead of a single caliper measurement taken in an arbitrary direction. And because of this, instead of examining each portrait to remark on mole here and a wart there in the subject, I am going to do a few lightning sketches from other perspectives.

The first step in understanding what standards for software documentation should be like is to understand where software, software documentation, and the use of software fit in the spectrum of human activities at work. This question has been studied by Rasmussen [1980], who has classified work activities as skill-based, rule-based, and knowledge-based. Since 1776 the distribution of human work among these categories has changed greatly.

Two hundred years ago most work was skill-based: The cabinet maker with his plane, the shepherd with his sheep made their decisions without conscious application of rules.

The Industrial Revolution mechanised many of these tasks, enabling only the most expert practitioners of skills to compete with machines. But it brought, instead, administrative work to manage the factories. This consisted of application of well-defined rules to easily recognized situations. By 1900 the need for people who could read rules and write decisions had become vital, and industrial societies were developing public education systems to provide universal literacy.

Since 1950, computing technology has been gradually making the average rule-based workers less employable, just as manufacturing technology did their skill-based great-grandfathers between 1850 and 1900. Nevertheless, many activities remain rule-based, even when this approach hinders efficiency -- as it does in the rule-based documentation for word processors, or when the rules create resentment -- as they do among medical staff used to giving rules rather than having rules imposed on them by software "assistants."

Ironically, although software may be used by people trained in rule-based work, the task of writing software is knowledge-based. By this I mean that it consists of determining facts, choosing hypotheses from a "knowledge base," pattern-matching them against the facts, and acting in ways determined by the results. The same is true of writing documentation. It is patently ridiculous to write standards in a rule-based style to

*This work was supported by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U.S. Department of Energy.

govern this activity. Unfortunately, we do not know how to write standards for knowledge-based activity. If we did, it would be as easy to teach style and semantics as it is to teach syntax. So much for the broadest brush in the paintbox.

What is a program? It is two things: an art and a contract. As an art it is "sculpture in thought constrained by reality," where correct syntax is necessary, but style and other more subtle things distinguish good from bad. Good software is art in contract draftsmanship, just as surely as Frank Lloyd Wright's houses are in brick and mortar. The architectural model of software leads to the following thought. Building codes set standards for housing construction. Are there similar functions in a set of standards for documentation? I am inclined to think not. Obviously, rules like those for Ph.D. theses about the use of 100% rag bond are ridiculous. Functional rules like "a roof shall be designed to withstand 150 pounds per square foot of static load, and an additional 100 pounds per square foot of wind load" seem attractive until we realise that, although it is easy to verify conformity with this standard, functional requirements for documentation are not verifiable in a way that would clearly stand up in court. I have written introductions to the use of full-screen text editors for our office staff and was pleasantly surprised by the success of a knowledge-based document where the knowledge base was set out with some care at the beginning by examples. But I have no idea whether this would work with office staffs in general.

Leaving the architectural model, let us look at software as a contract between users and computer. Before the contract can be drawn, the draftsman and the users must have a meeting of the minds about what is to be done. This is the requirements document, a "meta contract" or "meta program." Standards for requirements documents are thus "meta meta programs" -- hardly the subject for rule-based discourse.

After the contract has been drawn, it is like any other legal document -- incomprehensible to the user (but in our case, oddly enough, precisely comprehensible to the computer, apart from bugs). Therefore it needs an "interpretation" full of "for instances." This is the set of user manuals, which must meet three needs:

- It must be comprehensible to the user community. In particular, it must present facts about using the software, not facts from which these may be deduced.

- It must cover all commonly used cases and show the path to higher authority which can mediate unusual disputes between user and computer.

- Although it need not be precise, it must not be obviously misleading.

So much for the legal drafting and artistic models of software development and use.

COMMENTS

I will now add my own experience to some of the points raised by the other papers in this session. I think that such comments are worthwhile not only because the ideas are interesting but because the differences of emphasis help round out a picture of the process.

All documentation standards are context dependent. I can maintain the "C" programs that comprise UNIX with no other documentation than the source code, because Brian Kernighan will not let in the door anybody who has not read *The Elements of Programming Styles* [Kernighan and Plauger, 1978] (a policy that is a meta standard). If, on the other hand, I am maintaining the load modules of ADVENTURE which run on my Z80, I need a document in English to work with the program: The FORTRAN source is virtually incomprehensible; it seems to have been transliterated by hand or machine from some more suitable AI language, preserving only the instructions to the computer and omitting any explanation of the original writer's intent. There are issues here about correctly matching language to problems, and letting the compiler do the dog work, that suggest questions about transforming information from one representation to another. These issues are important; they are also an example of the general process that takes a product from a requirements document to a program, removing information about the application context and

adding information about the target computer system. By and large, documentation should preserve and extend the application dependent information stripped out of the requirements by a programmer, and should complement and explain the machine-dependent information inserted. This information which goes into the document depends on how much applications information is left in the program, how much more target machine information is needed to make the program do something desired by a user, and how much the user community may already be assumed to know.

Discussions of documentation as a QA tool make another important point. If you can't write a user manual because you can't explain how to use the program, then the program had better be rewritten by somebody who has at least read Kernighan and Plauger [1978]. The writer of user documentation needs the following things:

- No previous experience with the software being documented.
- The ability to explain clearly in writing.
- Means to obtain experience with the system to be documented by running it.
- A clear idea of what the users want the program to do.
- A document written by the programmer, believed to be a user manual, from which parts of the content of the user manual proper can be deduced.
- Access to the programmer to ask questions and, ideally, enough programming experience to be able to read and understand the source code.
- Authority to require a review by project management in cases where it seems that parts of the program should be rewritten.

I have seen the importance of these requirements underscored in my work as Chairman of the Board of Directors of a small software company (an activity I pursue in my spare time). We have found a good user manual to be our most effective marketing tool. This should imply that the Vice President for Marketing is an important ally to the documentation team. Similarly, the software team should be allies also, because a good user manual will save them field maintenance. But such allies require an understanding management, and such management is sometimes difficult to find.

Requirements documentation is important because it is the "draft agreement" between the users and the computer. If the final product does not meet requirements, and the need, was not documented there is bound to be a bitter dispute about who pays for necessary changes. Nevertheless, requirements documents often are not written at all, and those that are often are not very good. Two of the reasons are discussed below.

- The cost of a requirements document is "money up front." From the user's point of view this appears unnecessary risk capital, compounded by interest rates as the clock ticks until the project is finished. Low bidders, who promise to work without requirements, may seem far more attractive. If you want a customer to pay for a requirements document, you must be credible about the benefits and your ability to deliver them, right down to a better-than-average user manual.

Your most powerful marketing tools are horror stories about what happened to projects that were careless about requirements documentation. On such projects as nuclear electric generating plants, for example, the costs of slippage late in the project can be as high as a million dollars a day. Of course, deadlines invariably slip. If you are a good salesman and your customer is reasonable, you will be able to prepare for this kind of bad news, before you both sign a contract. (Unfortunately, such things happen only in the best of all possible worlds.)

- It is difficult to ensure that the requirements document contains the information needed. Two questions must be answered: "What is it for?" and "Who will use it?" As answers to these questions change, so too must the requirements document change. Thus a full requirements document has two parts: one a prospect, which is what we try to write

now, and the other a retrospect, which should be -- and almost never is -- written after all else is done.

Finally, documentation in general raises an AI theoretic issue. Most of us are familiar with the model of a finite-state machine which consists of nodes joined by paths, along which the incoming data stream causes transitions between the nodes. A generalisation of this seems to me to be a reasonable description of knowledge and, in fact, is one theoretical foundation for data bases that contain knowledge, i.e., knowledge bases. It is a set of networks, where all the items at about the same level of generality form a network that you can wander around using a process (i.e., program) to travel from node to node. The well-known ADVENTURE game is just such a network, although the process there is designed to give you a puzzle to solve rather than a guidebook. If you add to this network a feature where you can also descend vertically from each node into a network of more detailed information about that node, and make this feature indefinitely repeatable at will, you have a computer-based process that seems to me remarkably like the human process of mental information retrieval. As you descend into detail about some particular node, information about other unrelated topics at a higher level becomes less accessible in much the same way as nonrelated knowledge retires into the background mentally while we think intensely about one thing.

CONCLUSIONS

The notion of a documentation standard has been developed as a sort of box or pattern into which documents fit. The nature of the box is not yet properly defined to anybody. But it does seem close to ideas about representation of knowledge that have been developed by the AI community since the early 1970's [Woods 1970, Barr and Feigenbaum 1980].

And some things do seem clear. Documentation should meet the needs of several kinds of users. It should be in modules that can be "strung together," so that the same information is not written in several places. The master copy should be machine readable so that a computer can help with the inevitable updates. There may be several modules dealing with the same subject at different levels of detail. In such cases cases it should be easy to travel vertically through a set of more detailed modules, as well as horizontally from one module to another about something different at the same technical level. All of these travel processes should be easy to perform in documentation on paper, and should be facilitated on a computer by an intelligent process that remembers frequently used paths for any given user. The computer-based travel process should have a richer set of commands than the one on paper, which is probably restricted to UP, DOWN, and SIDEWAYS, speaking in metaphorical terms.

A final theoretical issue that seems to me to be important is the idea of "transformation." A program is obtained by a "transformation" of a "requirements document." The fact that the transformation is performed by people does not alter the central question of what information is discarded, what information is added, and what information should be moved to documentation. In fact, under this model the documentation is just as much a transformation of the requirements as the program is. The process of writing documentation consists of transforming the program, the requirements, and the description of the computing environment into a precis that covers most user needs and is not too heavy to hold! A standard for writing this precis has its theoretical foundation in a description of what kinds of knowledge should be preserved by the transformation and how to leave pointers in the precis to let the user find the material that was not preserved. Covering all needs is impossible; that is what the manuals for the computing environment, the customer's detailed procedures, and the program do together. In the case of dispute with the system not resolved by the documentation, these together with the computer system itself are the final court of appeal.

AN ACKNOWLEDGMENT AND A NOTE

The final text of this paper was prepared by Gail Pieper, using the notes from which I spoke as a draft. This gracious lady has firmly refused a co-authorship, and so the following acknowledgment takes its place. Thank you, Gail, for your part in the work here recorded.

Gail's effort in transforming my recollections into publishable form brought to the surface some thoughts I had been nursing about the relationship between software and documentation. Almost twenty-five years ago, when I wrote my first computer programs, it was a Herculean task to do anything, and explaining it in writing afterwards was trivial by comparison. Today I look at a large program, part of a research project by three colleagues which took eighteen months of part-time work for them to write. I do not think anybody will be able to document the system in detail to display its subtle yet clean and beautiful architecture without being an expert in several very difficult fields at once. The effort involved here in programming and documenting will be roughly equivalent, depending on where you put the tasks of analysis and specification.

This situation may easily become true of most programs. They will still be written by software artisans. But they will not be used without equal contributions from "wordsmiths," who have been using syntax and semantics to build perceptions ever since the first minstrel, long before Aristophanes. The wordsmith's task will have been recognised for what it really is: one leg of a tripod support for a software edifice, the other two being analytical insight into the problem at hand, and the art of program contract draftsmanship.

REFERENCES

Barr, A., and F. Feigenbaum, eds. [1980]. *The Handbook of Artificial Intelligence*, Vol. 1, Chapter 3. Stanford: Heuristech Press, pp. 143-222.

Kernighan, B. W., and P. J. Plauger [1978]. *The Elements of Programming Style.* 2nd ed. New York: McGraw-Hill.

Rasmussen, J. [1980]. "The Human as a System Component." *Human Interaction with Computers.* Ed. H. T. Smith and T. Green. Academic Press, New York. See also J. Rasmussen, "The Human Data Processor as a System Component. Bits and Pieces of a Model," Risoe-M-1722, 1974; "Notes on Diagnostic Strategies in a Process Plant Environment," Risoe-M-1983, 1978; "On the Structure of Knowledge. A Philosophy of Mental Models in a Man-Machine Context," Risoe-M-2192; and "Outline of Hybrid Models of Man and Machine," *Monitoring Behavior and Supervising Control*, ed. T. B. Sheridan and H. Johanssen, Plenum Press, 1976; and E. Holnagel, "A Framework for Description of Operator Behaviour," Risoe-N-35-79-NKA/KRU-P2(79)-24.

Woods, W. A. 1970. "Transition Network Grammars for Natural Language Analysis." CACM **13**: 591-606; see also W. A. Woods, "What's in a Link?" *Representation and Understanding*, Academic Press, 1975, pp. 35-82.

Session H: Quality Assurance for Documentation

Recommendations and Conclusions

Elisabeth F. Mullen

Managing Partner
JEM Associates
Herndon, Virginia

Following the presentation of papers which are included in these proceedings along with the discussant's remarks, there was a lively discussion among members of the audience and the panel. A summary of the issues raised, comments, and recommendations follows.

Issues raised by members of the audience included:

1. The relation between the Users' Manual and the Requirements Document: Should the first draft of the former be done from the latter?

2. Is the Users' Manual needed in order to complete testing?

3. Is there not a need for more than one type of Users' Manual?

4. What is known about the cost benefits of spending money on documentation? Is the investment returned in easier maintenance or greater reliability?

Comments on these issues included the following:

1. Caroline Levenson commented on the desirability of removing some of the duplication of effort implied by the current standard.

2. John Gabriel commented that the artificial intelligence community has tried to address the issue of different requirements for different groups of users and mentioned that there are some on-line documentation systems which use these ideas.

3. Herb Hecht commented that large systems tend to sink under the weight of their own changes. Often they must be rewritten because they can no longer be maintained. This is where a good requirements document is especially important.

4. John Gabriel pointed out the desirability of documenting the history of a system as well as its use. Examples included unusual origins for system routines and the need to replace computer systems which are part of systems with life times much longer than that of the computer hardware.

Recommendations emerging from the session include the following:

1. FIPS 38 does need to be evaluated.

2. Both the content and the context of the Users' Manual need to be reexamined in the light of current technology.

3. Several different types of users' manuals may be needed depending on the background and assignments of the users. Differentiating between the requirements of new and experienced users is desirable. The widely varying needs of the operations staff, maintenance programmers, and managers who must justify both the operating budget and the eventual replacement of the system should also be considered in determining both the number of types of manuals and their content.

4. The importance of considering the human interface was mentioned repeatedly, with a strong emphasis on providing enough flexibility in the standard to allow for some experimentation with new approaches in this area.

5. The project history may well be a valid subject for a full set of documentation.

6. There do not seem to be any hard facts on the incremental value of documentation, i.e. is the cost of documentation returned in increased maintainability or reliability? A study of this might be rewarding in terms of improved quality resulting from cost justifying the additional budget.

Concluding Session:

Summary of Findings and Recommendations

Leo Beltracchi

U.S. Nuclear Regulatory Commission

The concluding session of the workshop was moderated by Albrecht Neumann. Each session moderator of the previous sessions presented a summary of the findings and recommendations of their session. Comments and recommendations were also made by other members of the workshop. This provided an opportunity for all participants to hear and comment upon the findings of the workshop. A brief summary of the concluding session follows.

Alfred Sorkowitz, moderator of Session D, "Do Existing Standards Work?", presented the results of the session. These were:

1. FIPS PUB 38 is inadequate and should be revised. Examples of the subjects which need consideration are: software security, non-machine interactions.

2. To improve software documentation, it was recommended that technical writers be made a unique part of the design and development team. This would improve the importance of software documentation and the quality of the final product.

The results of a session survey on the use of documentation standards were also presented. The results indicated that organizations used in-house standards rather than a national standard for documentation.

Louis O'Korn of Session A, "Applying Documentation Standards - Case Studies" presented the results of the session. These were:

1. Existing software documentation standards provide good "How To" guidance. However, they are inadequate with regards to guidance on interactions by humans with computers, they are redundant in requirements, and they lack general guidance to the documenter.

2. New documentation standards are needed for all phases of software. Specific attention should be given to the operations and maintenance phase of the life-cycle and to the testing of software in the development phase.

3. More research is needed on documentation aspects for office automation. The issues of user implemented software, how to verify documentation and automated support tools for the non-computer professional must be resolved. Means should be found to publicize existing documentation guidelines, and standards.

4. It was noted a general lack of awareness of documentation standards exist.

5. The need for documentation must be defined in the initial work effort.

Raymond Houghton, moderator of Session C, "Tools for Improved Documentation", presented the following comments:

1. Documentation standards in hard copy have a tendency to become obsolete after some time. With a rapidly moving technology they need to be updated frequently, and must provide for introduction of new documentation techniques.

2. Some high level languages also provide human readable information which can augment other forms of documentation. This should be used in the overall systems development and systems design.

3. Documents and documentation are an integral and active part of an overall software life cycle. Software tools can play an important part in creating better and more cost effective documentation. Active management support is required and necessary to achieve these goals.

4. On-line documentation can serve as a "users manual". Split screen techniques can be used effectively. An integrated system of HELP messages, designed with several levels of generality also can provide useful user documentation. Interactive techniques, combined with split screen displays can be used effectively as user documentation.

5. Fully automated systems to perform user documentation for computer programs are not yet feasible at this time, and are 5 to 10 years away.

V. Douglas Hines of Session G, "Improving Human Interface", presented the following results:

1. Computer systems must be made more friendly to users. This will require user documents tailored to needs of each user audience.

2. New innovative approaches should be investigated to serve information needs of users, such as the use of color terminals, "comic book style" users manuals, and on-line documentation.

3. Terminal messages can be used as on-line documentation support for the user.

4. People must be trained to develop better documentation for computer systems.

5. Use software and systems documentation in the training of personnel.

Lenore Maruyama, moderator of Session F, "Enhancing Software Sharing", presented the following results:

1. Additional elements, such as a header label to record bibliographic information describing a machine-readable data file are needed.

2. FIPS 30 "Software Summary for Describing Computer Programs and Automated Data Systems" should be revised and updated.

3. As a tool for management, software sharing facilities can be used to save time and money.

Trudy Grieb, moderator of Session E, "Proposals for Documentation Standards", presented the following results:

1. A systems approach is needed to generate documentation. The issue of the cost and effectiveness of documents in terms of users needs and contract requirements must be resolved.

2. Documentation standards must be specific and definitive with regards to context, terms and concepts, but must be flexible to allow cost effective innovations.

3. Lessons learned from poor documentation should be recorded to avoid repeating the error. Also, the lessons learned from good documentation should be recorded to provide feedback and guidance to standard developers.

4. Because of the large consumer market, documentation for microcomputers may lead the effort for standards.

Virginia Walker, moderator for Session H, "Quality Assurance for Documentation", reported the following results:

1. FIPS 38 needs to be revised and updated.

2. Documents must strive to make the human interface with computers an easy one.

3. Management must recognize the role of documents in the life cycle of software such as the function of configuration control in program design and development.

4. Software documents must not only define what is recorded, but also the reason why it is recorded.

5. Documents for software must be logically oriented, well structured, and not fragmented.

At the conclusion of the summaries by the moderators, Al Neumann asked for comments from the floor. The following comments represent the major comments from the floor:

1. Electronic mail and teleconferencing can be used to develop standards and guidelines for software documentation. These methods allow for the participation of many people and reduces the need for periodic meetings.

2. We should strive to modularize documentation. Documentation for microprocessors is modularized and this accounts for its popularity. This concept eases the organization and control of document generation.

3. The need for documentation must be emphasized. This can best be achieved by defining the functional basis for the document. Software which does not have good documentation generally can not be trusted.

Upon termination of audience comments, Al Neumann thanked all participants for their contributions and then closed the workshop.

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBS SP 500-94	2. Performing Organ. Report No.	3. Publication Date October 1982

4. TITLE AND SUBTITLE

NBS FIPS Software Documentation
Proceedings of a Workshop
Held March 3, 1982, at NBS, Gaithersburg, MD

5. AUTHOR(S)

A. J. Neumann, Editor

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

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

7. Contract/Grant No.

8. Type of Report & Period Covered

Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

Same as item 6.

10. SUPPLEMENTARY NOTES

Library of Congress Catalog Card Number: 82-600600

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

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

These proceedings provide a record of papers and discussions presented at a workshop held on March 3, 1982, at the National Bureau of Standards. The meeting was sponsored by the NBS Institute for Computer Sciences and Technology. In addition to papers presented, the record also provides remarks by discussants and other participants. The workshop covered a variety of topics pertaining to software documentation. Topical sessions included: case studies of and reports on application of existing standards, documentation for operation and maintenance, tools for improved documentation, proposal for new documentation standards, enhancing software sharing, improving human interfaces, and quality assurance of documentation. Sixty-three papers were presented in parallel sessions, and a summary session concluded the meeting; over 300 persons participated in the workshop.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

documentation; FIPS; guidelines; program documentation; software documentation; standards.

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☒ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☐ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

294

15. Price

$8.50

USCOMM-DC 6043-P80

☆U.S. GOVERNMENT PRINTING OFFICE: 1982-360-997/2270

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SCIENCE & TECHNOLOGY

Superintendent of Documents,
Government Printing Office,
Washington, D. C 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Bureau of Standards Special Publication 500-.

Name _____

Company _____

Address _____

City _____ State _____ Zip Code _____

(Notification key N-503)

NBS TECHNICAL PUBLICATIONS

PERIODICALS

JOURNAL OF RESEARCH—The Journal of Research of the National Bureau of Standards reports NBS research and development in those disciplines of the physical and engineering sciences in which the Bureau is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Bureau's technical and scientific programs. As a special service to subscribers each issue contains complete citations to all recent Bureau publications in both NBS and non-NBS media. Issued six times a year. Annual subscription: domestic $18; foreign $22.50. Single copy, $4.25 domestic; $5.35 foreign.

NONPERIODICALS

Monographs—Major contributions to the technical literature on various subjects related to the Bureau's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NBS, NBS annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NBS under the authority of the National Standard Data Act (Public Law 90-396).

NOTE: The principal publication outlet for the foregoing data is the Journal of Physical and Chemical Reference Data (JPCRD) published quarterly for NBS by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements available from ACS, 1155 Sixteenth St., NW, Washington, DC 20056.

Building Science Series—Disseminates technical information developed at the Bureau on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NBS under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NBS administers this program as a supplement to the activities of the private sector standardizing organizations.

Consumer Information Series—Practical information, based on NBS research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

Order the **above** *NBS publications from: Superintendent of Documents, Government Printing Office, Washington, DC 20402.*

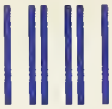
Order the **following** *NBS publications—FIPS and NBSIR's—from the National Technical Information Services, Springfield, VA 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NBS pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NBS Interagency Reports (NBSIR)—A special series of interim or final reports on work performed by NBS for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Services, Springfield, VA 22161, in paper copy or microfiche form.

U.S. Department of Commerce
National Bureau of Standards

Washington, D.C. 20234
Official Business
Penalty for Private Use $300



POSTAGE AND FEES PAID
U S DEPARTMENT OF COMMERCE
COM-215

SPECIAL FOURTH-CLASS RATE
BOOK