U.S. Department of Commerce
National Bureau of Standards

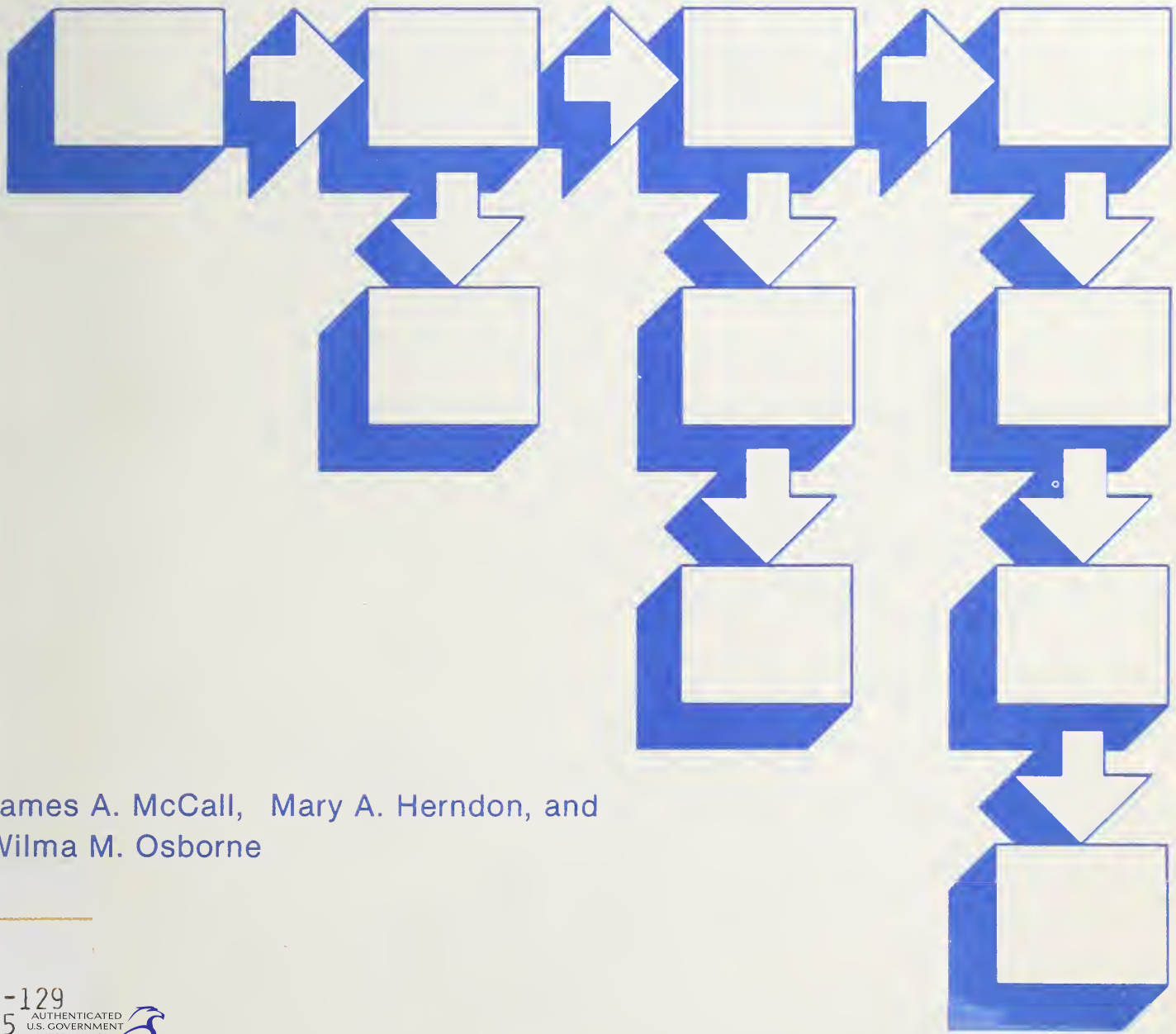
# Computer Science and Technology

NBS PUBLICATIONS

NAT'L INST. OF STAND & TECH R.I.C.
A11104 498310

NBS Special Publication 500-129

# Software Maintenance Management



James A. McCall, Mary A. Herndon, and Wilma M. Osborne

QC
100
U57
500-129
1985
c. 2
AUTHENTICATED U.S. GOVERNMENT INFORMATION GPO

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, the Institute for Computer Sciences and Technology, and the Institute for Materials Science and Engineering.

## *The National Measurement Laboratory*

Provides the national system of physical and chemical 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; provides advisory and research services to other Government agencies; conducts physical and chemical research; develops, produces, and distributes Standard Reference Materials; and provides calibration services. The Laboratory consists of the following centers:

- Basic Standards[2]
- Radiation Research
- Chemical Physics
- Analytical Chemistry

## *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

## *The Institute for Materials Science and Engineering*

Conducts research and provides measurements, data, standards, reference materials, quantitative understanding and other technical information fundamental to the processing, structure, properties and performance of materials; addresses the scientific basis for new advanced materials technologies; plans research around cross-country scientific themes such as nondestructive evaluation and phase diagram development; oversees Bureau-wide technical programs in nuclear reactor radiation research and nondestructive evaluation; and broadly disseminates generic technical information resulting from its programs. The Institute consists of the following Divisions:

- Inorganic Materials
- Fracture and Deformation[3]
- Polymers
- Metallurgy
- Reactor Radiation

[1]Headquarters and Laboratories at Gaithersburg, MD, unless otherwise noted; mailing address Gaithersburg, MD 20899.
[2]Some divisions within the center are located at Boulder, CO 80303.
[3]Located at Boulder, CO, with some elements at Gaithersburg, MD.

NATIONAL BUREAU OF STANDARDS LIBRARY

# Computer Science and Technology

NBS Special Publication 500-129

# Software Maintenance Management

James A. McCall and Mary A. Herndon

Science Applications International Corporation
La Jolla, CA 92038

Wilma M. Osborne

Center for Programming Science and Technology
Institute for Computer Sciences and Technology
National Bureau of Standards
Gaithersburg, MD 20899

Issued October 1985




**U.S. DEPARTMENT OF COMMERCE**
**Malcolm Baldrige, Secretary**

**National Bureau of Standards**
Ernest Ambler, Director

## 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: 85-600596
National Bureau of Standards Special Publication 500-129
Natl. Bur. Stand. (U.S.), Spec. Publ. 500-129, 65 pages (Oct. 1985)
CODEN: XNBSAV

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1985

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402

ABSTRACT

This report focuses on the management and maintenance of software and provides guidance to Federal government personnel to assist them in performing and controlling software maintenance. It presents an overview of the various aspects of software maintenance including the problems and issues identified during the ICST sponsored survey of Government and private industry maintenance organizations. Techniques, practices, tools, and procedures which aid in reducing these problems and which help to insure that quality software is developed for and by the Federal ADP community are identified. A definition of software maintenance is provided which recognizes that maintenance includes enhancing a system to meet users needs, modifying a system to adapt to a changing environment, as well as correcting errors in the system. An integrated approach to software maintenance is described with suggestions for improving the software maintenance process. These suggestions provide a basis for improving both the software quality and the productivity of the organization.

Keywords: cost control measures, decision aids, management, software configuration management, software maintenance management, software maintenance tools, software quality assurance, test plans.

TABLE OF CONTENTS


| Section | | Page |
|---|---|---|
| 1.0 | EXECUTIVE SUMMARY | 1 |
| 2.0 | BACKGROUND | 3 |
| 2.1 | Purpose | 3 |
| 2.2 | Introduction | 4 |
| 2.3 | Organization of Report | 6 |
| 3.0 | SOFTWARE MAINTENANCE DEFINITION | 8 |
| 3.1 | Software Maintenance as a Phase of The Lifecycle | 9 |
| 3.2 | Software Maintenance Characteristics | 11 |
| 3.3 | Software Maintenance Problems | 12 |
| 4.0 | MANAGING THE MAINTAINER | 16 |
| 4.1 | The Difficulty of Software Maintenance Activities | 16 |
| 4.2 | The Maintenance Career Path | 17 |
| 4.3 | Attributes of Maintenance Personnel | 17 |
| 4.4 | Personnel Scheduling | 19 |
| 4.5 | Deadline Assignments | 19 |
| 4.6 | Performance Criteria and Personnel Effectiveness | 20 |
| 5.0 | THE MAINTENANCE ORGANIZATION AND USER INTERFACE | 21 |
| 5.1 | Establishing the User Interface | 21 |
| 5.2 | User Interface and Task Prioritizing/Scheduling | 22 |
| 6.0 | ECONOMIC CONSIDERATIONS OF SOFTWARE MAINTENANCE | 24 |
| 6.1 | Cost Control Measures | 24 |
| 6.2 | Cost Estimation Models | 24 |
| 6.3 | Work Breakdown Structures | 25 |
| 6.4 | Specifying Resource Requirements: A Software Maintenance Management Plan | 25 |
| 7.0 | SOFTWARE MAINTENANCE TECHNIQUES | 30 |
| 7.1 | Problem Reporting Procedures | 30 |
| 7.1.1 | Forms | 33 |
| 7.1.2 | Data analysis | 33 |
| 7.2 | Software Quality Assurance Procedures | 33 |

TABLE OF CONTENTS (cont)


| | | |
|---|---|---|
| 7.3 | Configuration Management | 34 |
| 7.4 | Test Plans and Procedures | 37 |
| 8.0 | MAINTENANCE SUPPORT TOOLS AND TECHNIQUES | 39 |
| 8.1 | Types of Tools | 39 |
| 8.1.1 | Support librarians | 39 |
| 8.1.2 | Code analysis | 39 |
| 8.1.3 | Requirements database and requirements tracing | 39 |
| 8.1.4 | Test data management | 40 |
| 8.1.5 | Test and error analysis | 40 |
| 8.1.6 | Code instrumentors/execution analyzers | 40 |
| 8.1.7 | Text editors | 41 |
| 8.1.8 | Structure/restructuring | 41 |
| 8.1.9 | Application generation | 41 |
| 8.1.10 | Simulation/emulation | 41 |
| 8.1.11 | Management tools | 42 |
| 8.2 | An Integrated Maintenance Environment | 44 |
| 8.3 | Build or Buy Decision | 44 |
| 9.0 | MAINTENANCE MANAGEMENT DECISION AIDS | 46 |
| 9.1 | Redesign Guidance | 46 |
| 9.2 | Structured Versus Unstructured Coding Techniques | 47 |
| 9.3 | Regression Test Coverage | 47 |
| 9.4 | Requirements Impact Analysis | 48 |
| 9.5 | Major Versus Minor Modification Assessments | 48 |
| 10.0 | SOFTWARE MAINTENANCE PROBLEMS TO AVOID | 49 |
| 10.1 | Common Mistakes and Pitfalls to Avoid | 49 |
| 10.2 | A Strategy for Implementing Survival Techniques | 51 |
| 11.0 | PERFORMANCE GOALS OF SOFTWARE MAINTENANCE MANAGEMENT | 52 |
| 11.1 | System Availability and Reliability | 52 |
| 11.2 | User Satisfaction | 52 |
| 11.3 | Change Request/Problem Report Response Time | 53 |
| 11.4 | Productivity | 53 |

TABLE OF CONTENTS (cont)


12.0 SUMMARY AND CONCLUSIONS . . . . . 54

13. REFERENCES . . . . . 59


LIST OF TABLES


| Section | | Page |
|---|---|---|
| 2.3-1 | TOPIC INDEX | 7 |
| 3.1-1 | SOFTWARE MAINTENANCE DEFINITION HIERARCHY | 8 |
| 3.1-2 | MAINTENANCE ACTIVITIES IN EACH CATEGORY | 10 |
| 3.2-1 | SURVEY RESULTS | 15 |
| 4.3-1 | ATTRIBUTES OF MAINTENANCE PERSONNEL | 18 |
| 5.1-1 | KEY INFORMATION ON SOFTWARE CHANGE REQUEST | 23 |
| 6.4-1 | OUTLINE OF A SOFTWARE MAINTENANCE MANAGEMENT PLAN | 28 |
| 6.4-2 | IMPLEMENTING A SOFTWARE MAINTENANCE MANAGEMENT PLAN | 29 |
| 8.1-1 | TOOL TYPES | 43 |
| 8.2-1 | TOOL PRIORITIES | 44 |
| 9.2-1 | REDESIGN TRADEOFFS | 46 |
| 10.1-1 | COMMON MISTAKES AND PITFALLS | 50 |
| 12.1-1 | IMPROVING THE MAINTAINABILITY OF SOFTWARE | 56 |
| 12.1-2 | MAINTENANCE CONSIDERATION DURING DEVELOPMENT | 57 |
| 12.1-3 | ATTRIBUTES OF MAINTAINABLE SOFTWARE | 58 |


List of Figures


| | | |
|---|---|---|
| 6.2-1 | Example of Workflow | 26 |
| 7.1-1 | Problem Reporting Process for Maintenance | 31 |
| 7.1.1-1 | Sample Problem Report Form | 32 |
| 7.2-1 | SQA Code Walkthrough Checklist | 35 |
| 7.3-1 | Flow of System Problem Reports and Change Request Through Configuration Management | 36 |

# 1 EXECUTIVE SUMMARY

Software maintenance is a critical function within most large organizations. The reliability of software systems upon which these organizations depend is the software maintainer's mission. This mission includes not only correcting errors, but modifying the system to make it more effective for the user. In spite of its importance to the organization, software maintenance often has been ignored as a significant management concern.

This report focuses on the management of software maintenance. Specific problems, identified during a survey of Government and private industry software maintenance organizations are described. These problems fall into the following categories:

- The level of resources required for software maintenance;
- The lack of management discipline and visibility into the maintenance process;
- The lack of formal techniques and decision aids for performing software maintenance;
- The lack of support tools for the maintainer;
- Personnel issues such as turnover, career paths, and assignment and training techniques; and
- The legacy of development - the poor quality of the software and documentation that must be maintained.

These problems make it difficult for organizations to: meet user's changing needs; effectively manage a large inventory of application software; control the increasing costs of maintenance; and keep up with the demand for new application software development.

A definition of software maintenance is provided which recognizes that maintenance includes enhancing a system to meet users needs, modifying a system to adapt to a changing environment, as well as correcting errors in the system. An integrated approach to software maintenance is described with suggestions for improving the software maintenance process.

Our recommendations are that software maintenance can be improved by taking the following steps:

- o DEVELOP A SOFTWARE MAINTENANCE PLAN
- o RECOGNIZE IMPROVEMENT OF MAINTAINABILITY QUALITY ASSURANCE)
- o ELEVATE MAINTENANCE VISIBILITY IN THE ORGANIZATION
- o REWARD MAINTENANCE PERSONNEL, PROVIDE A CAREER PATH AND TRAINING
- o FORMALIZE QUALITY ASSURANCE AND TEST PROCEDURES
- o ESTABLISH AND ENFORCE STANDARDS
- o RECORD ALL SYSTEM CHANGES (KEEP DATA)
- o INTRODUCE SOFTWARE TOOLS
- o USE MODERN PROGRAMMING TECHNIQUES
- o CONTROL DATA (INCLUDING TEST DATA)
- o USE SCHEDULED MAINTENANCE CYCLES

Additional improvements can be realized by introducing software maintenance considerations during the early phases of a software system's life cycle. In order to develop maintainable software products, the goal of maintainability must be established during the requirements, design, and development stages and built into the software using standards, and proven design and development techniques. It is essential, however, to involve the managers, maintainers, and users during each of these stages.

## 2 BACKGROUND

This report was produced in support of the National Bureau of Standards Institute for Computer Sciences and Technology (ICST) acting in response to its Brooks Act charter to promote the cost effective selection, acquisition, and utilization of automatic data processing resources within Federal agencies. ICST efforts include research in computer science and technology, direct technical assistance, and the development of Federal standards for data processing equipment, practices, and software.

As part of this responsibility and the need to improve software maintenance methods and management, the ICST is developing software maintenance guidance designed to assist Federal agencies in the ongoing support of existing computer systems. While software systems vary in function, type, and size, many of the functions performed under software maintenance are universal in scope and the activities required to keep these systems operational are generally the same.

This report, which is part of a comprehensive family of software maintenance guidance, addresses management and technical practices and discusses the need for a maintenance policy with enforceable controls for use throughout the software lifecycle.

### 2.1 Purpose

The purpose of this report is to provide guidance to Federal government personnel to assist them in performing and controlling the software maintenance process. It presents an overview of the various aspects of software maintenance, including the current, most pressing problems. It identifies techniques, practices, tools, and procedures which aid in reducing these problems and which help to insure that quality software is developed for and by the Federal ADP community.

The objectives of this report are:

- to provide a set of definitions relating to software maintenance for use by Federal agencies. Define the various aspects of software maintenance;
- to describe a methodology which provides the information and guidance necessary for the preparation and implementation of disciplined software maintenance procedures;

- to provide specific and detailed guidance for managing software maintenance;

- to provide specific and detailed guidance on the various techniques and methods which produce more effective software maintenance management practices;

- to provide guidance on techniques and tools that can be used during original development to reduce the cost and difficulty of software maintenance;

- to provide an analysis of the problems associated with software maintenance; and

- to help Federal agencies to produce higher quality software and reduce software costs.

While this report is prepared specifically for use within the Federal government, it should be applicable and useful to software maintenance managers and practitioners in industry as well.

## 2.2 Introduction

It is currently estimated that software maintenance accounts for sixty to seventy percent of each software dollar allocated [NBS106], [MART83, GAO81b, LIEN76]. Software, as it is used here, refers to computer programs, data, and documentation. In some environments, it is reported that programmers spend up to eighty percent of their time on maintenance functions. It is important to note that over the lifecycle, software maintenance consumes a far greater amount of resources than software development and inevitably costs more. Software maintenance is also a highly visible, labor intensive activity subject to both application area and computing environment changes. If software maintenance is to be improved, aid is needed to improve not only the quality of the software, but the environmental factors as well. This will require the adoption of improved maintenance techniques, tools, procedures, and a management philosophy which incorporates planned and anticipated or preventive software maintenance.

A number of Federal Agencies have initiated measures to help achieve more effective software maintenance. The U.S. General Accounting Office (GAO) produced several reports addressing the cost and problems associated with software maintenance and the need for guidance in this area. The first report, [GAO80], suggested that unless Federal agencies make more use of modern software tools and techniques for development and maintenance, Federal computer software will continue to cost

millions more than is necessary. The second report, [GAO81a], concluded that Federal data processing systems often are not cost effective, do not meet user needs, have cost overruns and simply do not work. This was attributed in part to inadequate management control during system development and inadequate control over software changes. In a third report [GAO81b], it was estimated that two-thirds of the programmers on fifteen sites visited by the GAO, spent their time on maintenance. The National Bureau of Standards (NBS) SP 500-106 entitled "Guidance on Software Maintenance" identifies software maintenance problems and provides recommendations for reducing both its difficulty and cost.

One of the comments frequently encountered in the literature is that maintenance costs are difficult to control because of the disparity in how maintenance is defined. There is also a difference of opinion on whether all of the costs attributed to maintenance are justified. Many contend that the majority of what is termed software maintenance is in fact, on-going development. For what is considered development in one environment is considered maintenance in another. Thus, a comprehensive set of definitions which encompasses all of the functions commonly identified as maintenance is needed. This would facilitate better software management and a more even assignment of maintenance costs.

Lientz and Swanson [LIEN80] defined software maintenance to include all modifications made to an existing applications system, including enhancements and extensions. They divided software maintenance into three major categories: corrective, adaptive, and perfective. The GAO document AFMD-81-25 [GAO81b] defineds maintenance to include: the removal of defects, tuning the software for efficiency and economy, modifying the software for end-user satisfaction, and adapting the software to new operating conditions. Most of the managers surveyed agree that maintenance generally refers to those activities required to keep the software system operational from the time it is accepted until the software is replaced. Regardless of how maintenance is defined, many of the same activities need to be performed. If maintenance is to be performed effectively, then a methodology which employs the best available techniques and tools must be used.

## 2.3 Organization Of Report

This section provides an overview of the structure of the Guide and includes an alphabetized table indexed by topic.

Section 1 provides an executive summary, highlighting the general observations and guidance offered in the report.

Section 2 provides the background, introduction, purpose, and organization of the report.

Section 3 provides a standard definition for software maintenance and summarizes the current problems faced by most maintenance organizations.

Section 4 discusses the types of qualifications and experience needed in a maintenance environment.

Section 5 discusses user interface responsibilities in a software maintenance environment.

Section 6 examines current software maintenance costs, describes estimating techniques and the concept of a software maintenance management plan.

Section 7 describes specific maintenance techniques ranging from problem reporting, quality assurance, configuration management, testing, and standards and conventions which should be employed as standard practices.

Section 8 provides an overview of software support tools which can be beneficial in a software maintenance organization.

Section 9 describes decision aids that can be helpful to managers or supervisors within software maintenance organizations.

Section 10 identifies common mistakes and pitfalls and discusses how to avoid them.

Section 11 recommends performance goals that will assist in monitoring and evaluating the organization's maintenance activities, and provides details on how to develop maintainable software.

Section 12 provides the summary and conclusions.

TABLE 2.3-1 TOPIC INDEX


| TOPIC | SECTION NUMBER |
|---|---|
| Common Mistakes/Pitfalls | 10 |
| Configuration Management | 7 |
| Costs | 6 |
| Decision Aids | 9 |
| Definitions | 3 |
| Estimating | 6 |
| Executive Summary | 1 |
| Introduction | 2 |
| Maintenance Measures | 11 |
| Management Plan | 6 |
| Organization | 5 |
| Performance Evaluation | 4, 11 |
| Performance Goals | 11 |
| Personnel Issues | 4 |
| Problems | 3 |
| Problem Reporting/Change Requests | 7 |
| Quality Assurance | 7 |
| Redesign Criteria | 9 |
| Scheduling | 4 |
| Software Maintainability | 11 |
| Software Support Tools | 8 |
| Standards and Conventions | 7 |
| Task Assignments | 5 |
| Testing | 7 |
| Test Coverage | 9 |
| Workbreakdown Structure | 6 |

# 3 SOFTWARE MAINTENANCE DEFINITIONS

This section provides a constituent set of software maintenance definitions and establishes a common basis for discussion in this report. This set of definitions is in the form of a hierarchy. At the highest level, software maintenance is defined as a phase of the lifecycle of a software product. At the next level, the different types of maintenance typically performed are identified. This is the level at which managers are likely to relate to from a functional standpoint. At the lowest level, the activities or generic tasks typically performed during software maintenance are defined. The concept of this hierarchy of definitions is shown in Table 3.1-1.

In the first two levels of the definition hierarchy, the works of Swanson [SWAN76] and Martin and Osborne [MART82] have been used extensively.

TABLE 3.1-1 SOFTWARE MAINTENANCE DEFINITION HIERARCHY

| | |
|---|---|
| Software Maintenance: Performance of those activities required to keep a software system operational and responsive to its users after it is accepted and placed into production. | LEVEL 1<br>---<br>Definition as a phase of lifecycle |
| Perfective<br>Adaptive<br>Corrective | LEVEL 2<br>---<br>Categories of maintenance |
| Requirements Analysis<br>Design Analysis<br>User Interface<br>Design Review<br>Configuration Control<br>Code Audits<br>.<br>.<br>. | LEVEL 3<br>---<br>Maintenance activities |

## 3.1 Software Maintenance As A Phase Of The Lifecyle

The lifecycle of a computer software system is defined as the period from its initial conception until it is no longer used. Traditionally, the lifecycle phases have been defined as requirements, design, implementation, testing, and operation and maintenance. These phases are generally associated with the development of a software system. Software maintenance, however, is usually associated with those activities performed after development. Thus, the definition of software maintenance as a lifecycle phase is:

> the performance of those activities required to keep a software system operational and responsive to its users after it is accepted and placed into production.

There is a set of products associated with each lifecycle phase. These products encompass requirements and design documentation, code, the data base, users manuals, test plans, procedures and results, operators manuals, etc. Software maintenance, then, is a set of activities which results in changes in the baseline product.

### CATEGORIES OF SOFTWARE MAINTENANCE

Maintenance activities are frequently divided into three categories which were originally proposed by Swanson [SWAN76]. These are: perfective, adaptive, and corrective. In this Guide, these categories are defined as follows:

- Perfective Maintenance

  All changes, insertions and deletions, modifications, extensions, and enhancements made to a system to meet the evolving and/or expanding needs of the user are considered to be perfective maintenance. Activities designed to make the code easier to understand and use, such as restructuring or documentation updates (often referred to as "preventive" maintenance) are considered to be perfective as well as optimization to make the code run faster or use storage more efficiently. Estimates indicate that more than 60% of all maintenance effort falls into this category.

- Adaptive Maintenance

  Adaptive maintenance consists of all effort initiated as a result of changes in the environment in which a software system must operate. These environmental changes are normally beyond the control of the software maintainer and consist primarily of changes to the computer hardware, operating system, operating system tools (compilers, utilities, etc.) and terminal devices. Note that changes in requirement specifications by the user are considered to be perfective, not adaptive. Estimates indicate that approximately 20% of all maintenance effort falls into this category.

- Corrective Maintenance

  Changes necessitated by actual errors (induced and residual "bugs") in a system are considered to be corrective maintenance. This category consists of activities normally considered to be "fire-fighting" or "quick fixes" to allow a system to continue to be operational. Corrective maintenance is usually a reactive type process where an error must be fixed immediately. Estimates indicate that only 20% of all maintenance effort falls into this category. Table 3.1-2 provides some example activities of each of these categories.

TABLE 3.1-2 MAINTENANCE ACTIVITIES IN EACH CATEGORY

| CATEGORY | ACTIVITY |
|---|---|
| PERFECTIVE | Making Code Easier to Understand |
| | Improving Documentation |
| | Optimizing the Code |
| | Adding a New Capability |
| ADAPTIVE | Modifying application for new version of operating system, compiler, peripherals, or DB |
| CORRECTIVE | Quick fixes and fire-fighting |

## 3.2 Software Maintenance Characteristics

Software maintenance involves many of the same activities associated with software development. One way of describing the activities of software maintenance is to identify them as successive iterations of the first four phases of the software lifecycle, i.e. requirements, design, implementation, and testing. Software maintenance is this successive iteration with unique characteristics of its own. The unique characteristics include the time-frame in which maintenance is performed, the personnel performing maintenance, the environment of the software product, and the testing process. Software maintenance is typically performed within a much shorter time-frame than a software development effort which often spans one or more years.

With respect to personnel, software development projects are usually staffed with a specific mix of skills to meet the requirements of the application and development efforts. People with different skills are assigned at the various phases of the development to take full advantage of their expertise. In contrast, all maintenance activities for a particular software product may be performed by one person, acting as requirements analyst, designer, coder, and tester. Although this is not always the case, the separation and assembling of a certain mix of skills is more typical during development than during maintenance.

Another characteristic of maintenance is that the maintainers usually are not the same persons who developed the system. Thus, the maintainers must analyze and understand the existing product before they can modify or correct the software, usually without any inherent knowledge of the implementation strategies employed by the developer. While this is a subtle difference, it is significant. It is also the basic reason people traditionally have stated that maintenance is not a creative process, motivating many 'good' programmers to seek development projects. Since maintenance activities are performed within the context of an existing framework or system, it is generally thought that an individual's flexibility is somewhat constrained. This, more than any other factor, presents the most challenging problem for maintenance personnel. Moreover, the older the system, the more difficult the system is to maintain.

Software maintenance can be viewed as successive iterations of the development phases, but its uniqueness should be recognized to insure effective management, staffing and planning considerations. The following is a list of functions typically performed while maintaining software:

Change Requirements
Impact Analysis
User Interface
Problem Report Recording and Control
Change Request Recording and Control
Configuration Control
Data Base Modification
Code Modification/Recompilation
Code Debugging
Module/Subsystem/System Testing
Documentation Modification
Code Inspections/Walkthroughs
Test Data Generation
Management Planning and Control
Field Delivery
Administrative Support
User Assistance/Training

Depending on the environment, a certain subset of the these activities may be associated with corrective maintenance while another subset may be associated with perfective maintenance. The processes are influenced by such factors as the size of the change, response time or other time constraints, whether the maintenance is performed on-site or at a centralized maintenance facility, etc.

## 3.3 Software Maintenance Problems

Recognition of the cost of software maintenance to the Government was the catalyst for the focus of attention on maintenance. Two surveys, one by the General Accounting Office (GAO) and one by Lientz, Swanson, and Tomkins, contributed significantly to this recognition. Each survey identified the major problems in software maintenance. Table 3.2-1 summarizes the findings of these two surveys along with the findings of the survey conducted by the National Bureau of Standards (NBS). The problems identified in each of these surveys are grouped according to six major problem areas.

The first problem area identified is that of cost. GAO estimated that two-thirds of the programming staff in the Federal government are involved in maintenance. Statistics from Department of Defense studies report up to 75% of the lifecycle budget for software systems is related to software maintenance [RADC82]. A recent book on software maintenance [MART83] estimates $30 billion are spent annually on software maintenance world wide. These types of statistics have only recently been recognized due to the lack of a standard definition of what constitutes maintenance and to the difficulties in obtaining cost accounting and management data.

Management problems are considered to be the second most significant area of concern. The problems in this group involve managing/recognizing maintenance as a separate function; user and upper level management's perception of maintenance; a lack of goals, standards, and criteria to judge performance; and managing the user interface. These problems can be attributed to the fact that unique aspects of software maintenance were not recognized in the past.

The third problem area specifically addresses technical approaches (techniques) and procedures for performing software maintenance. While there are many standards, methodologies, techniques, and procedures advocated for software development, there are few associated with maintenance. Emphasis on software development problems is a familiar theme within the software research and software engineering communities. However, until recently software maintenance has received little attention. A typical response to the reason for this neglect is that if software is developed well in the first place it will be easy to maintain. This response ignores the fact that sixty to seventy percent of maintenance consists of modifying software to perform new functions or to comply with new requirements. It also ignores the sizeable software inventory currently being maintained that was developed without the use of modern programming practices. Without disciplined approaches to maintenance, these systems are characterized by an exponential growth in size and complexity during their post delivery life. The result is generally premature system demise or instability and progressively more costly maintenance.

The fourth problem area identified in the survey is software tools. There are few tools available which directly apply and support the unique aspects of software maintenance. The lack of specific tools for software maintenance is compounded by the fact that tools common to development organizations that would also benefit software maintenance are rarely transferred to the maintenance organization.

The fifth problem area involved personnel issues such as availability, lack of training, turnover, motivation, experience, scheduling, etc.

The last problem area identified by maintenance personnel is the legacy of development. This legacy is that software turned over to the maintainers is often poorly designed, poorly implemented, and poorly documented. Thus, the maintainer has a poorly structured product to maintain without the proper information to understand it. These attributes all contribute to making the software more difficult to fix or modify.

The existence of these problems in a data processing organization can result in:

- an inability to meet the user's changing needs;
- an inability to manage effectively a significant investment in a large inventory of software programs and data;
- an application backlog, where most resources are devoted to maintaining existing systems, preventing development of new applications. (The costs of not developing a new system are typically not accounted for but can be significant.); and
- a decline in the quality of systems because of poor software maintenance, resulting in more expensive future maintenance.

Table 3.2-1 Survey Results

| Problem Areas | GAO Survey [GAO 81b] | Lientz and Swanson [LIEN 78] | NBS Survey [MCCA 83] |
|---|---|---|---|
| Cost | o No cost accounting or management data available<br>o 2/3 of programmers in Federal Government | o Budgetary problems<br>o Over 50% of DP staff | o 50%-80% of DP personnel budget<br>o Budgetary restrictions on staff positions and funding |
| Management | o Not managed as separate funotion<br>o No widely accepted definition<br>o No goals, standards, or criteria for good performance | o User demands for enhancements<br>o Schedule commitments<br>o Lack of user understanding<br>o Unrealistic user expectations<br>o Forecasting personnel requirements | o Upper managements perception of maintenance |
| Formal Techniques | | o Changes to hardware and software environment<br>o Adherance to programming standards<br>o Data integrity | o Lack of control procedures<br>o Frequency of changes to environment |
| Tools | o Limited use of tools | | o Lack of available tools which support maintenance |
| Personnel | o Software often maintained by different people than those that developed it | o Competing demands for people<br>o Availability of maintenance personnel<br>o Turnover<br>o Skills of maintenance personnel<br>o Motivation of personnel | o Availability of maintenance staff<br>o Turnover<br>o Preception of maintenance career path |
| Legacy of Development | o Poor documentation | o Quality of documentation<br>o Quality of original programs<br>o Adequacy of design specs | o Quality of documentation<br>o Quality of code<br>o Quality of design<br>o Antiquated system |

# 4 MANAGING THE MAINTAINER

The software development process has evolved to a set of procedures that are performed in a prescribed order. Although performance of the procedures often requires several passes before the desired quality is obtained, the course of action is generally better defined than the maintenance process. Although the maintenance process can be viewed as iterations of the development process, there are unique aspects of maintenance which seemingly have no procedural counterpart to development. There are numerous training courses on software development techniques, but only a few corresponding types of courses that address software maintenance. While numerous studies have investigated the inherent difficulty of constructing a complex system, only a few studies have addressed the difficulty of maintaining a complex system. Furthermore, the results of these studies have contributed to the development of tools and techniques to aid in the tasks of software requirements definition, design, coding, and testing. Maintenance tasks are, in general, less organized than development tasks. The situation is further complicated by the fact that documentation and other information necessary to complete maintenance tasks are often unavailable.

## 4.1 The Difficulty Of Software Maintenance Activities

One research study [CIRA71] related maintenance to trying to find "a needle in the haystack". For example, when a software error occurs, how does the analyst proceed to isolate, analyze, and repair the problem? There are no tried and true techniques that can immediately isolate the routine at fault. Rather, the analyst must perform some "detective work" based upon analytical skills and perseverance, and must be able to reconstruct the processing scenario at the time the problem occurred. The process of reconstructing the scenario can be extremely difficult, involving complex processing. The analyst must decide, on the basis of clues, which parts of the system contain the faults. This process, sometimes referred to as "over confirmation of clues" [CIRA71]), is inefficient in terms of the number of logic paths the analyst must trace before identifying the faulty one.

Examination of the psychological processes necessary for incorporating a major enhancement, reveals further sources of difficulties. For example, upon beginning a task, the analyst may have a reasonable idea of what routine will be affected. However, before that intuition can be verified, the impact on the design must be assessed. This assessment necessitates a

jump upward in the level of abstraction. If the affected module is tightly coupled with other modules, additional jumps upward are necessary. After identifying all of the modules implicated, the analyst must move quickly to a lower level of abstration, (i.e. the code) to begin modifications. Studies have shown [CIRA 71] that the process of hopping from one level of abstraction to another is error-prone and inefficient.

Since maintenance managers and technical personnel are judged by their success in meeting deadlines, there is often an attempt to hurry to complete the tasks. Pressure mounts and mistakes occur. Thus, the physical environment may not be conducive to maintaining the concentration level essential for the task.

## 4.2 The Maintenance Career Path

Throughout the industry, the role of the maintenance staff has not always been viewed favorably. Often, a person is assigned to a maintenance activity after performing development tasks poorly or simply as an "entry-level" assignment. Careers are sometimes short-lived because of the lack of management support and a good organizational attitude about software maintenance. The very nature of maintenance responsibilities can result in a demanding 24 hour on-call environment. In some cases, the personnel must agree to be available on weekends in the event of a system emergency. Fortunately, the traditional viewpoint and other unfavorable conditions are disappearing in the DP community.

Organizations should identify a specific career path for maintenance personnel. This path may be identified in the same terms as development personnel, i.e. progression from an entry-level programmer to a senior programmer to an analyst to a supervisor to a unit manager. However, the key is to demonstrate that there is a career path.

## 4.3 Attributes Of Maintenance Personnel

Software maintenance is a critical function in most organizations; thus, it is important that the staff be highly skilled. The key attributes of successful maintenance personnel are highlighted in Table 4.3-1. Attempts should be made to hire persons with these attributes; they should also be encouraged in the existing staff through training courses and by managers who reinforce them during meetings and counseling.

TABLE 4.3-1 ATTRIBUTES OF MAINTENANCE PERSONNEL

| ATTRIBUTES | TECHNICAL BACKGROUND |
|---|---|
| EXPERIENCED | Application<br>Coding/Debugging<br>Testing |
| TECHNICAL PROGRAMMING SKILLS | Understand existing software<br>Ability to make software more maintainable |
| ABLE TO RELATE TO USERS NEEDS | |
| ABLE TO RELATE TO OPERATIONAL ENVIRONMENT | |
| VERSATILE | Knowledge of various hardware, operating systems, languages. |
| ANALYTICAL | Able to find error.<br>Able to analyze impact of change. |
| ADAPTABLE | Able to work under sometimes poor conditions, using poor documentation. |
| RESOURCEFUL/ IMAGINATIVE | Must be creative, (i.e. must be able to diagnose problem, create work around or invent a method for enhancing a system within existing constraints). |
| OBJECTIVE | Must recognize deadline and technical constraints of task. |
| THOROUGH: | Fixes and changes must be thoroughly debugged and tested. |

## 4.4 Personnel Scheduling

A frequently practiced management strategy in software maintenance is to assign a "resident expert" to every system that is maintained. While the simplicity of this management strategy is attractive, the overall effectiveness of the maintenance team may be reduced. Depending upon the size and urgency of the request load, bottlenecks may develop, especially if the resident expert is unavailable. Another result of this approach is an eventual deterioration of the documentation because the resident expert fails to maintain it. In fact, undocumented changes enhance the maintenance expert's image and job security. If this resident expert leaves the organization, however, the system is left in an unmaintainable state. One way to avoid this type of disaster, is to make multiple system assignments so that no single individual becomes the weak link in the chain.

Another approach is to rotate assignments so that the lack of one person will be less of a bottleneck in the completion of tasks. An added advantage of rotational assignments is the increase in the responsibility level of personnel. The concept of rotational assignment is also applicable between development and maintenance assignments.

## 4.5 Deadline Assignments

Maintenance activities typically are more susceptible to deadlines than development activities. Problems that result in a system crash demand an immediate response, e.g. banks must post transactions on a daily basis or incur both a loss of funds and the wrath of customers. Similarly, payroll offices are beset with complaints if weekly payroll checks are not on time. Inevitably, the maintenance tasks of "keeping the system up and running" generate the pressing deadline requirements for the technical staff. While the ability to meet deadlines is often an important evaluation criteria for the technical staff, missed deadlines reflect negatively on management as well.

Effective techniques for forecasting resource requirements are often not available. Thus, task completion estimation techniques for software consist of heuristic recipes rather than replicable analytical techniques. If the technical staff is unduly rushed in performing maintenance tasks, there is a high probability that the quality of work will be lower. In the process, maintenance standards and procedures are often ignored in an effort to "get it out the door". The end result is degraded system and software quality and more deadline assignments.

The tendency to "rush to fix" tasks can be reduced by:

- o rotating "on-call" assignments among the staff so no one person is relegated to continuous on-call assignment. This may be more difficult to accomplish in small organizations than in large ones;
- o offering incentives to lessen the psychological burdens;
- o estimating the resources required for enhancements and other major modifications before assignment;
- o taking the complexity of the change and the system into consideration when estimating the time required for change implementation; and
- o considering the presence or absence of adequate documentation.

## 4.6 Performance Criteria And Personnel Effectiveness

Accurate and objective personnel evaluation techniques for programmers and analysts are not well established in the industry. Programmer productivity, whether in a maintenance or a development setting, has proven to be difficult to measure. While many types of measures have been proposed, (lines of code produced per day, success in meeting deadlines, cost per line of code, etc.) no one set of measures has been unanimously accepted as a standard. Furthermore, programmer productivity takes on a separate meaning when considering maintenance rather than development.

Maintenance productivity metrics should evaluate the criteria that reflect the measure of satisfaction to the user and organization as a whole. Some examples of appropriate metrics include the ability to meet a deadline (if realistically set); the degree of quality of the maintenance activities (post-maintenance problem report frequency); the ability to improve future maintenance by preparation of adequate documentation; and the observation of standards and procedures.

# 5 THE MAINTENANCE ORGANIZATION AND USER INTERFACE

The historical perspective is that maintenance is a necessary evil, required because the software was not developed well in the first place. This attitude is changing rapidly in today's environment where computers and software are present in all facets of our life. Almost every major organization is dependent to some extent on the effective performance of software maintenance. Thus, an understanding of the difficulties, challenges, and requirements of software maintenance by manager and users is essential.

There are a number of factors which contribute to efficient and cost effective software maintenance. Most of these factors are discussed in detail in sections six through eleven of this Guide. The two factors that are perhaps the best indicators of how well a software maintenance organization is performing, however, are effective user interface and user satisfaction.

## 5.1 Establishing The User Interface

Effective user interface helps insure that all persons involved or affected by the change request are aware of each others' requirements, as well as the functional capabilities of the system. It is one of the most important responsibilities within a maintenance organization. Most maintenance (approximately 60%) is performed specifically at the users request (to modify the system). The user interface group would have responsibility to:

- Forecast User's Needs/System Changes;
- Complete System Change Request;
- Estimate time and resources requirements;
- Review Estimates;
- Keep informed of status;
- Participate in tests;
- Review User's Manual;
- Understand User's situation, people, needs; and
- Educate user to the environment, organizational capabilities, feasible responses.

User interface should be established with a degree of formality and conducted on as friendly a basis as possible. The degree of formality can be introduced by the use of a system change request form. This form not only introduces some formality, it provides an historical record of system changes and provides a basis for managing the change process. Table 5.1-1 identifies the key information the change request form should contain. A more detailed example of a problem reporting

form which could also be used for change requests may be found in section seven of this report. The requestor should always provide as much information as possible about the cause of the problem or the nature of the change. This will not only assist the maintainer in changing the software, and but will help management better assess the cost of making the change.

## 5.2 User Interface And Task Prioritizing/Scheduling

When possible, fixed maintenance cycles should be established. The user should be involved in the assignment process. An effective technique is to have periodic user group meetings where new and current system change requests, availability of resources, and priorities can be reviewed. It is important that the user recognize that the maintenance organization is a limited resource and that change requests must be prioritized to effectively utilize that resource. Organizations that employ scheduled maintenance cycles are able to manage their resources more effectively than those that react to daily ad hoc demands.

TABLE 5.1-1 KEY INFORMATION ON SOFTWARE CHANGE REQUEST FORM

- o Identification of Requestor
- o Date of Request
- o Change Desired (description and why)
- o Date Change Needed/Priority
- o Analysis (description of approach to change)
- o Identification of Resources Required
- o Responsible People
- o Schedule/Milestones
- o Date Change Made (with maintainer and user sign off)

# 6 ECONOMIC CONSIDERATIONS OF SOFTWARE MAINTENANCE

The effort required to maintain systems developed within the last five years is typically less than that required for older systems. The reason is that these systems typically have been developed using structured high level languages, structured design and implementation methodologies, well documented system utilities and data base management systems. As a result, these systems tend to be easier to understand and modify.

## 6.1 Cost Control Measures

Maintenance organizations are beginning to recognize the need to improve systems as they are modified. Evidence suggests that a 20% to 50% reduction in future maintenance costs can be achieved by the consistent use of software quality standards, tools, and techniques (See [MCCA83] and [MART83].).

Another cost control measure that has proven effective, though not always compatible with an organization's accounting procedure, is the use of a charge back system. In a charge back system, the users pay for maintenance on the applications that support them. The advantage of this system is that requests for modifications are evaluated in detail , and in terms of their cost/benefits by persons responsible for the cost. A related benefit is that the user gains insight into the maintenance process. While it is difficult to determine the cost savings provided by a charge back system, the use of this technique has the effect of encouraging more efficiencies in the system by users and maintenance staff.

## 6.2 Cost Estimation Models

Most cost estimation models are parametric models for which the user provides input parameters describing the system in terms of size, application, language, complexity, etc. Some of these models provide an estimate of lifecycle costs for the system.

The output of the models is the estimated level of effort/cost to develop and maintain the system. This output can be used to estimate the cost of making a major change to an existing system. The change may be described to the model in terms of the amount of new code to be developed and the percentage of the existing code to be modified. Further information can be found in [THIB81] and [COOK80].

## 6.3 Work Breakdown Structures

A work breakdown structure is a technique used for planning and controlling projects by decomposing each project into manageable work units. This technique is used often when Government organizations contract for development or services. Essentially, this technique requires a planner to describe the workflow within the maintenance organization. For a particular maintenance task, this work flow can be used to plan/estimate resources required and milestones. An advantage of using a consistent work breakdown structure across many tasks is that planning and estimating become easier and more accurate because of the empirical data upon which those plans are based. Figure 6.3-1 provides an example of a work flow plan. Each box represents a work unit that could be associated with a work breakdown structure accounting code.

## 6.4 Specifying Resource Requirements: A Software Maintenance Management Plan

The mode of many software maintenance organizations responding to problem reports and change requests is both ad hoc and reactive. It is difficult to project resource requirements on a long term basis using this mode of management. We recommend that a plan be developed to assist management in specifying project resource requirements. This plan should provide a brief description of the application system including the hardware/software environment on which the production system runs. Any variations or multiple configurations should be defined. Organizations which use the system or have responsibility for its development and maintenance should be identified. Their responsibilities and interfaces should be outlined.

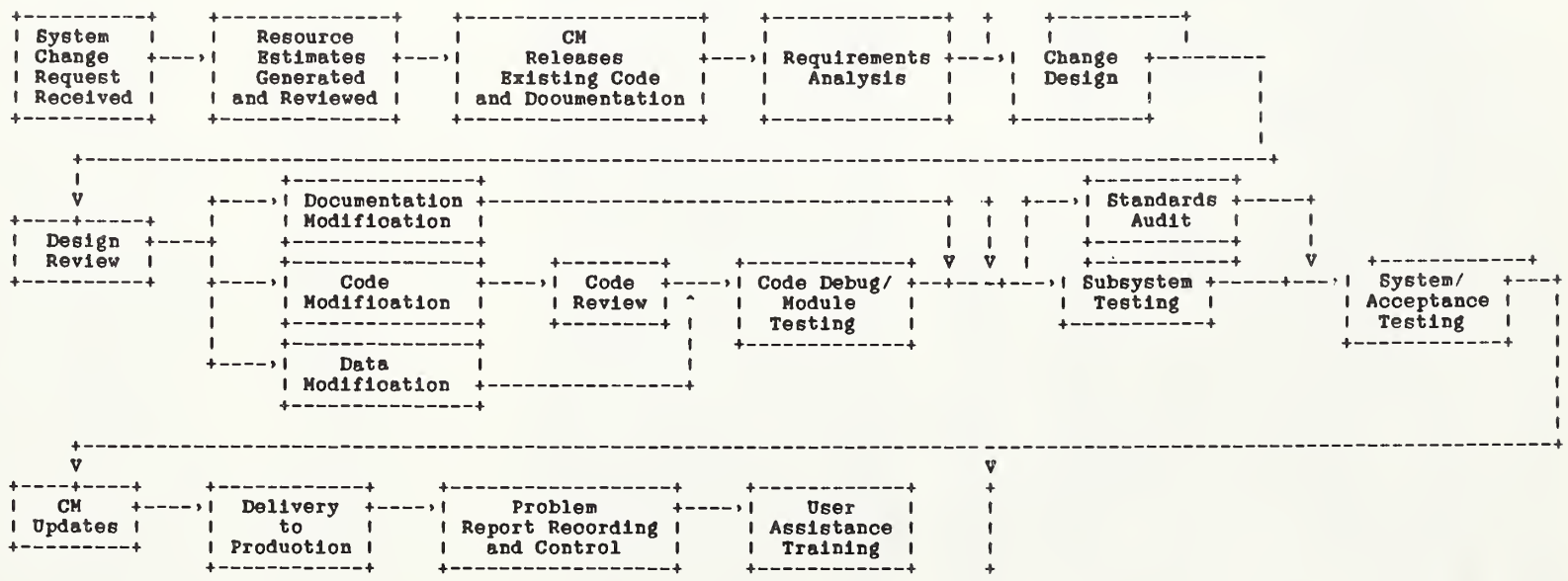



Figure 6.2-1
Example of Workflow

The plan should outline the procedures to be followed in maintaining the systems. The procedures should address how system changes are requested by the user and how priorities and schedules are to be assigned. Other procedures to be covered include problem reporting by the user, provisions for customer assistance, and response times for problem resolution, provisions for review, approval, and implementation of configuration and version changes, and any planned maintenance cycles. Any quality assurance activities and testing activities (regression testing) including user involvement in acceptance testing should also be addressed.

The plan should identify such resource requirements as hardware, software, personnel, facilities, and supplies. Care should be taken to include the anticipated time requirements for maintenance of the system. The plan should also identify support software tools and methods to be used in maintaining the system.

The plan may change as the situation changes or as lessons are learned and feedback obtained. The fact that it exists, however, provides continuity, formality, and a rationale for the procedures in place. The result of these influences on the maintenance organization is an introduction of a more disciplined approach to maintenance providing a more effective resource to the overall organization. Table 6.4.1 describes key attributes of a software maintenance management plan; while Table 6.4-2 descibes specific steps that should be taken when implementing a software maintenance management plan.

TABLE 6.4-1 OUTLINE OF A SOFTWARE MAINTENANCE MANAGEMENT PLAN

1.0 Introduction
- Purpose
- Scope
- Applicable Documents

2.0 System Description
- Hardware (Application system Production Environment)
- Software
- Definitions/Glossary

3.0 Management
- Identification of User, Developer, Maintainer
- Responsibilities
- Interfaces

4.0 Procedures
- System Change Requests/Problem Reports
- Customer Assistance
- Configuration Control
- Quality Assurance
- Testing/Acceptance Testing
- Maintenance Cycles

5.0 Resources
- Hardware(Maintenance Environment)
- Support Software
- Personnel (Assigned Responsibilities)
- Facilities
- Supplies

TABLE 6.4-2 IMPLEMENTING A SOFTWARE MAINTENANCE MANAGEMENT PLAN

- Inventory all application software and documentation.
- Establish maintenance management plans for each application.
- Establish a common workflow.
- Assign personnel to key functions such as configuration management, quality assurance, user interface, and testing.
- Establish standards and conventions.
- Initiate the use of system change request and problem report forms and maintenance logs.
- Document a plan to purchase or build support tools incrementally. Present this plan to upper management for approval.
- Establish procedures for fixed maintenance cycles to handle user change requests and problem reports (critical problems in a high priority mode).
- Establish a training program including in-house training, on-the-job training, and commercial seminar/course attendance.
- Brief upper level management periodically on the progress of the improvement plan citing statistics collected from problem reports, change request forms and maintenance logs.
- Schedule periodic meetings with staff to solicit ideas on effective techniques or tools.
- Communicate with developers and provide them with feedback on maintenance problems, including problem type and frequency. The goal of this exercise is to improve software development standards and quality.

# 7 SOFTWARE MAINTENANCE TECHNIQUES

Effective control of a software maintenance organization usually results in a higher quality product. Controlling the activities performed by the maintenance personnel is best accomplished when there is a definitive set of procedures to follow. The techniques and procedures discussed in this section include: problem reporting (forms usage, data analysis), software quality assurance, code walkthroughs, software configuration management, and test plans and procedures.

## 7.1 Problem Reporting Procedures

A critical aspect of user satisfaction is a responsive problem reporting procedure. Problems can be residual (delivered with the system) or introduced as a part of modification activities. Regardless of which type of problem exists, maintenance organizations must be prepared to catalogue problems efficiently for either future or immediate analysis and correction. An example of a useful reporting system is a data base management system that maintains the current problem reports and their status, catalogues the problem reports, and provides various error analysis as depicted in figure 7.1-1. Key aspects of a problem reporting procedure include:

- a form for the user to fill out;
- discussion of the problem report with the user and estimate of the time and resources required to correct the problem;
- documentation of the analysis of the problem;
- categorization the problem for future use;
- involvement of the configuration control function in assigning the priority and insuring the update is made properly; and
- documentation the completion of the fix.

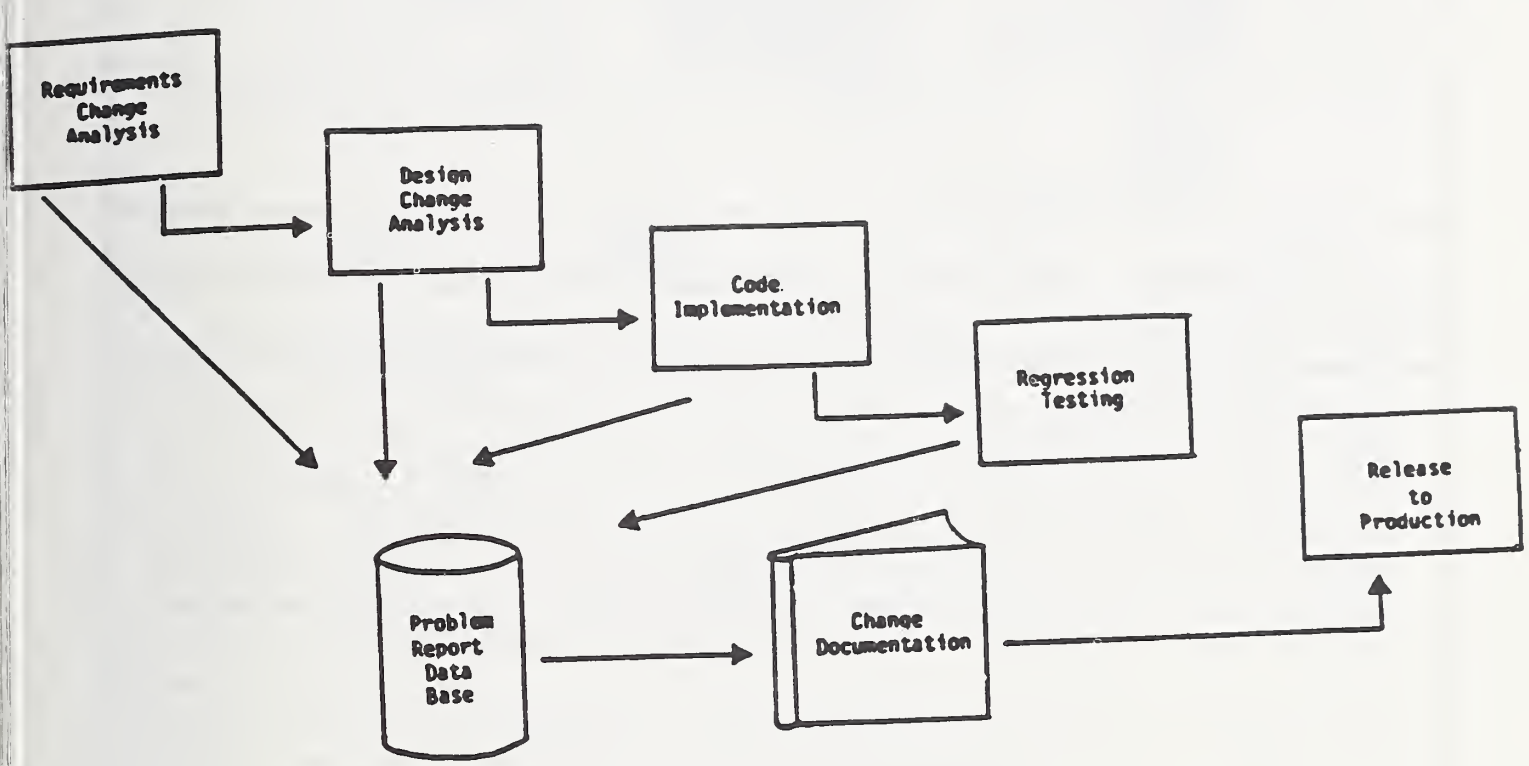



Figure 7.1-1 Problem Reporting Process for Maintenance

TITLE: _____ NUMBER: _____

_____ DATE: _____

PROGRAM ID: _____ ANALYST(S): _____

REFERENCES: _____

_____

_____

HEADING: (circle one heading, plus one subheading when furnished)

Computational
- missing equation
- division by zero
- ambiguous statement
- erroneous calculation
- unnecessary calculation
- mixed mode

Data Handling
- subscript errors
- argument list inconsistencies
- uninitialized variable
- data transfer errors
- data structure errors
- scaling/precision errors

Logic Flow
- missing test
- erroneous
- wrong sequence

Documentation
- erroneous
- insufficient
- ambiguous
- violation of coding standard

Design

Tester

CRITICALITY LEVEL: (circle one) Low, Performance Substandard — Medium, Performance Unacceptable — High, System Stops

METHOD OF DETECTION: _____

DESCRIPTION OF PROBLEM: _____

EFFECTS OF PROBLEM: _____

RECOMMENDED SOLUTION: _____

Circle one:
- U = OPEN
- S = CLOSED
- C = CORRECTED, CLOSED
- U = UNSUPPORTED, CLOSED

REQUIREMENTS LINKS NUMBERS: _____

CLOSED DATE: _____

Figure 7.1.1.1 Sample Problem Report Form

### 7.1.1 Forms -

Problems can originate from the field, from users, or from in-house analysts and programmers. Often, the need to retrieve information quickly is very important in maintaining high user-satisfaction. A form that records certain useful descriptive information can aid in this task. Figure 7.1.1-1 contains an example of a Problem Report Form. Problem reports vary in form but the information content is generally as shown. More advanced problem reporting systems, based on data base management systems can assist in analysis and provide statistics on the types and frequency of problems being reported.

### 7.1.2 Data Analysis -

The data gathered by the problem reporting system can be used to describe the changing quality of the system. For example, the project manager may wish to examine each module in a system for a frequency distribution of the number of problems by module. Experience has indicated that sources of errors are rarely uniformly distributed across all modules. Rather, a subset of modules will exhibit high error rates. Once the problem areas are pinpointed, management is better able to determine whether it would be more cost effective to redesign rather than continue maintaining these modules.

## 7.2 Software Quality Assurance Procedures

The software quality assurance (SQA) procedures should identify the documents that will be reviewed, the personnel involved, and the schedule. These procedures and responsibilities should be documented in a software quality assurance plan. The SQA procedures should, as a minimum, involve design and code walkthroughs. They may also include a documentation audit, test monitoring, and an SQA sign off prior to release.

The most common of these procedures is the code walkthrough which is performed typically by a group of four people. The author of the code explains the program statement by statement. During this time, a checklist is used by the moderator to eliminate common coding errors. The checklist identifies such items as: data item initializations, subscripts in array references, pointer updates, computation errors and comparison errors. An example checklist is shown in Figure 7.2-1. As the walkthrough progresses, group interaction results in a trouble shooting analysis of the software. One benefit of the group

interaction is that design errors are often discovered by other team members.

## 7.3 Configuration Management

The primary objective of software configuration management (SCM), generally referred to as the management of software changes, is the release of operationally correct, reliable, and cost effective software. SCM consists of four functions: identification, configuration control, status accounting, and auditing. It helps to ensure that software changes satisfy specified requirements and test criteria. It is also the responsibility of SCM to: provide for records retention, disaster recovery, library activities, a software repository, and to ensure that the necessary coordination and approvals are obtained prior to changing the baseline. SCM helps to track all actions associated with a problem report or change request. Figure 7.3-1 identifies the flow of control for problem reports and change requests using SCM.

| Code Walkthrough Checklist | Inspector: |
|---|---|
| System Name:<br>Program Name: | Date: |

STRUCTURE

1. Does the program exceed established size standards?
2. Does the program have only one entrance and one exit?
3. Does the processing flow from top to bottom?
4. Does the number of decisions exceed established complexity standards?
5. Does the number of paragraphs exceed established standards?

DOCUMENTATION/COMMENTS

1. Are there prologue comments that identify function, inputs and outputs, variables, author, modifications made, limitations, etc?
2. Are decisions commented?
3. Are variables described by comments?
4. Are branches commented?
5. Is all machine language code commented?
6. Do comments do more than repeat operation?
7. Is consistent indentation and spacing used?

DATA

1. Are all variables names unique?
2. Does each variable have only one unique name?
3. Are variables used in only one way?
4. Are global variables used consistently with respect to units and type?
5. Are all elements of an array/table functionally related?
6. Are all variables initialized before use?
7. Are all default values described?
8. Are arrays/tables/strings initialized before use?

INTERFACES

1. Are all calls to other programs commented?
2. Are all arguments within calls parametrics?
3. Does the calling program maintain control?

ERROR HANDLING

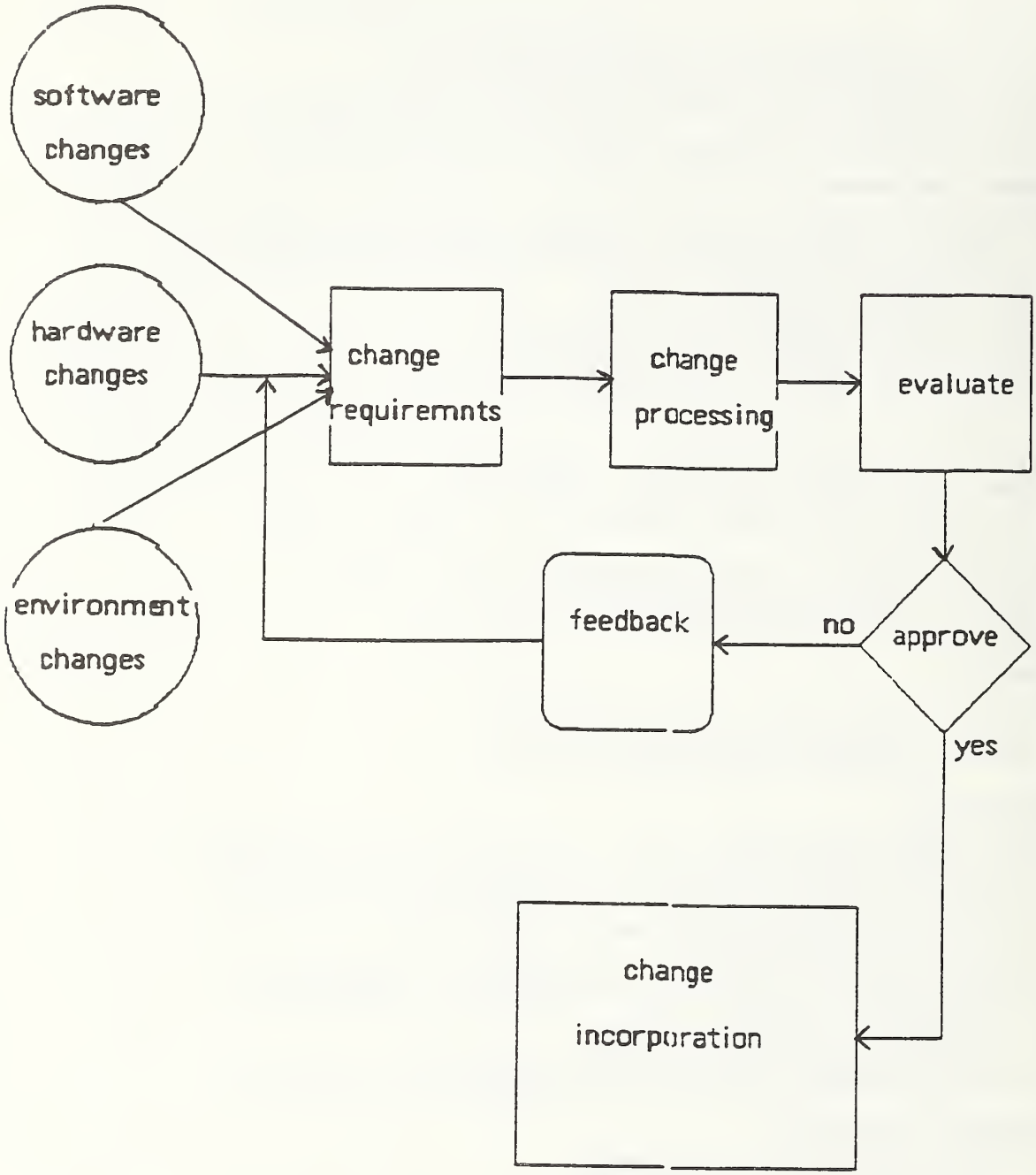
1. Are inputs range tested?
2. Are possible conflicts or illegal combinations of inputs checked?
3. Is there a check to determine if all necessary data is available prior to processing?
4. Are loop and branch index parameters range tested?
5. Are subscripts range tested?
6. When an error condition occurs, is it passed to a calling module?
7. Are the results of a computation checked before outputting or processing continues?
8. Are there any mix made expressions?
9. Are all unusual termination conditions described?
10. Are error messages descriptive and necessary actions explained?

CONSISTENCY/COMPLETENESS

1. Does the code represent the design?
2. Are all global variables defined?
3. Are all called programs defined?
4. Is all code reachable?
5. Are all labels necessary?
6. Are nonstandard language features avoided?
7. Is self-modifying code avoided?

Figure 7.2-1 SQA Code Walkthrough Checklist

Figure 7.3-1 Flow of System Problem Reports and Change Requests Through Configuration Management



## 7.4 Test Plans And Procedures

Effective test plans and procedures can help to ensure software quality during maintenance. The test plan should be established with milestones and estimates for each step that indicate when a system fix or modification is to be designed, coded, tested, and placed into production. A reasonable amount of time should be allocated for testing prior to release. Management should also require that status reports, which provide visibility into the testing schedule, be received promptly from the person or team responsible for making modifications and testing. Otherwise, intermediate milestones may slip without any adjustment being made to the final milestone or release date. This often results in a compressed testing schedule which makes thorough testing impossible. There should be a standard approach for testing and using automated aids for test case generation and storage. The standard approach should include the following steps:

- o plan for enough time to thoroughly test a modification to a system;
- o require the development and use of test cases which thoroughly test the software;
- o archive the test data for future regression testing;
- o prepare tests that are based on user scenarios and on the functional requirements of the system; and
- o establish a set of criteria for test coverage that provides confidence that the testing has been thorough. This set of criteria may include such procedures as:
  - a percentage of all lines of code must be executed;
  - a percentage of all decision paths within the software must be executed; and
  - execution of all error handling processes in the code.
- o Use a parallel test system.
- o Involve user in acceptance of system.

There should be a commitment to establish and maintain a standard set of test cases (data) for each application. This set of test data can be used for regression testing whenever software modifications are required.

While there are many software tools that support the test process, many maintenance programmers develop their own tools to assist in debugging and testing code. It is recommended that a standard set of tools be used in the test process. Tools to be considered range from test data generators, to test database management systems, to code instrumentors/execution analyzers. The following NBS publications provide additional information in the areas addressed in this section: [FIPS106], NBS106], [FIPS101], [NBS98], and [NBS93].

# 8 MAINTENANCE SUPPORT TOOLS AND TECHNIQUES

During the survey of maintenance organizations, most managers expressed a desire for more software support tools. Many were not aware of the existence of tools for maintenance activities, Others stated that their budget was not sufficient to purchase additional tools. The following section identifies the types and function categories of the tools that are available. A more detailed description of these tools may be found in Table 8.1-1 and in the NBS Special Publications 500-88 [HOUG 82].

## 8.1 Types Of Tools

### 8.1.1 Support Librarians -

There is a wide variety of tools which control changes to the configuration or version of source code. They range from specific configuration control systems to programmer utilities which facilitate standard program construction and modification processes. Most maintenance organizations practice configuration control but only about half use an automated tool to support that activity. This is an area in which automated support librarian tools are especially useful.

### 8.1.2 Code Analysis -

One of the first steps a maintenance programmer performs, when debugging a reported error is to examine the source code. Considerable time is spent looking at the source code since the documentation is often out of date. Code analysis tools can be used to process the source code for: standards enforcement, data flow analysis, complexity analysis, generation of cross reference listings, or production of flow charts.

### 8.1.3 Requirements Database And Tracing -

When a user requests a change to a function or capability of the system, the change is not identified with a particular routine. In a large system, tracing a change (requirement) through the design of the system to where it is implemented can be time consuming. Tools exist that provide this type of traceability. In some cases these tools require the specifications to be written in a formal language ([ALFO77], [TEIC77]) and in other cases they do not [HERN81]. In all cases, these tools can be very advantageous to the maintainer

attempting to analyze how to modify a system and what impact a change will have.

### 8.1.4 Test Data Management -

Test data may be prepared manually, generated from actual production data sets, or prepared as a result of analysis of the code. Once prepared, the test data should be maintained for regression testing. Tools such as test data generators and data base management systems support these activities and provide considerable cost savings by preventing the maintainers from generating and storing test data manually.

### 8.1.5 Test And Error Analysis -

Problem reports from the users are valuable sources of information to a maintenance organization. They not only identify corrections that must be made but help to keep track of the frequency, number, and type of problems being reported, enabling an organization to improve its performance over time. This improvement can be realized by completely redesigning and rewriting programs which exhibit high error rates, by assigning senior personnel to those problem programs, and by instituting standards which will alleviate occurrence of certain types of errors during modifications or correction activities.

### 8.1.6 Code Instrumentors/execution Analyzers -

Tools exist which instrument source code with counters, sensors, and assertions, and then provide reports after test cases have been run against the code. These tools report code and path coverage and assertions during the test/execution run providing some assessment of the thoroughness of testing. Other tools in this category assist in optimization by identifying frequency of execution of individual statements and segments of the code. Such tools support the debugging process by allowing a programmer to step through the execution of a program during execution. These types of tools provide automated support in analyzing the dynamic behavior of the code, and are particularly helpful when tuning a system or during testing (regression testing) changes to the system.

### 8.1.7 Text Editors -

Text editors are used extensively in software environments to produce documents and to enter and modify code. A significant benefit can be realized in maintenance if the documentation produced during development is delivered to the maintenance organization in machine readable form. Modification (updates and revisions) of the documentation then becomes a process similar to that of code modification and is more easily enforced upon the programmers. Keeping the documentation up to date can provide significant savings in debugging and designing modifications.

### 8.1.8 Structuring/restructuring -

Several tools have emerged in the market place in the last few years which are designed to generate structured code from unstructured code. The benefits of working with a structured product can be significant to a maintenance organization. In most cases, the inventory of unstructured code must be considerable to warrant the expense of a restructuring tool. Other tools in this category include structured language preprocessors to facilitate structured coding and language conversion packages which support conversion of code.

### 8.1.9 Application Generation -

Application generators are high level languages oriented toward a specific type of application which allow use of functional statements to build a system. In the past, these types of tools have not facilitated maintenance because of the difficulty in debugging, tuning, and testing in the high level language, the lack of support tools, and the scarcity of personnel experienced in application generators. However, recent advances in these types of tools make them more accommodating to the maintainer. Their use is dictated to the maintainer if the development was performed using one.

### 8.1.10 Simulation/emulation -

Simulations and emulations are typically developed for a specific application. They are used to assess performance of the system. In maintenance, they can be used to assess whether performance goals are met after modification. They also are useful if an application is being maintained on a computer system which is not the same as the production system. In this

case, a simulator or emulator allows testing to be done on the maintenance facility prior to fielding the new version.

### 8.1.11 Management Tools -

There are a number of tools which can assist in managing a maintenance organization. Tools such as schedulers can be used to keep track of different tasks, personnel assignments, milestones, etc. Tools that utilize data from various sources can provide management data which describe the amount of change being made to a system, the growth in complexity of that system as a result of the changes, the performance of the maintenance organization in terms of user problem reports, and response time for completion of change requests. Although these tools may not increase the maintenance staff's productivity, they do assist the manager in making better decisions about resource allocation.

TABLE 8.1-1 TOOL TYPES

| TOOL TYPES | FUNCTION CATEGORIES IN NBS SPECIAL PUBLICATION 500-88 [HOOG 82] |
|---|---|
| Support Librarians | formatting, configuration management |
| Code Analysis | cross-reference, data flow analysis, structure checking, comparison, interface analysis, type analysis |
| Requirements Data Base and Tracing | completeness checking, consistency checking, tracability |
| Test Data Management | test data generation, regression testing |
| Test and Error Analysis | problem report data base. |
| Code Instrumentation/ Execution Analysis | coverage analysis, tracing, tuning, symbolic execution, assertion checkin debugging support |
| Text Editors | editing |
| Structure/Restructure | translation, restructuring |
| Application Generators | synthesis |
| Optimizers | optimization |
| Simulations/Emulations | simulation |
| Management Tools | schedulers, PERT/CPM, status reports, cost estimation models |

## 8.2 An Integrated Maintenance Environment

A recommended approach to realizing long term benefits from software tools is to accumulate an inventory of tools that will support the various activities performed by the maintainer. The goal should be to establish an integrated set of tools which support a logical sequence of activities (a methodology), share data, and provide a common user interface. Table 8.2-1 provides a list of tools grouped by the order in which they should be accummulated to maximize phasing in the tools under budgetary constraints. This concept can have tremendous benefit to an organization if all personnel are using the same set of tools.

TABLE 8.2-1 TOOL PRIORITIES

| | |
|---|---|
| First Phase | Compiler (structured languages) |
| | Text Editor |
| | Configuration Manager/ Support Library |
| | On-line Debugger |
| Second Phase | Code Analyzer |
| | Test and Error Analysis |
| | Optimizers |
| | Code Instrumentation/ Execution Analysis |
| | Simulation/Emulation |
| | Management Tools |
| Third Phase | Restructurer |
| | Requirements Data Base |
| | Application Generator |

## 8.3 Build Or Buy Decision

Software support tools have always suffered from the "not invented here" syndrome. Programmers often prefer to develop their own utilities and tools rather than use existing ones. Traditionally, tools have been poorly documented, very specialized, and difficult to move from one environment to another. There has been significant improvement in commercial tool development, and many of the software tools now offered are reliable, and well documented. Therefore, a build or buy decision process should be conducted prior to building a tool

in-house. If a tool that sufficiently meets the requirements of the maintenance organization can be purchased commercially, it is usually cost beneficial to do so rather than building one in-house.

# 9 MAINTENANCE MANAGEMENT DECISION AIDS

Managing a maintenance organization is a challenging task. Decision aids assist management by helping them decide when to redesign rather than to continue making patches; when structured coding techniques should be used on unstructured code; and what the impact of a change will be on the system and on resource requirements.

## 9.1 Redesign Guidance

One of the more difficult questions that arises during change analysis, is when to redesign. The basic question is whether a change to a system should be simply a modification to the code (a patch) or if a particular segment (module or modules) of the system should be redesigned and rewritten. Table 9.2-1 provides the basic tradeoffs involved in making this decision.

TABLE 9.2-1 REDESIGN TRADEOFFS

| Why Patch | Why Redesign |
|---|---|
| o Expediency | o Feasibility of change |
| o Schedule | o Efficiency of Performance |
| o Effort | o Restructure Code |
| o Continued operation | o Use modern design/code techniques |
| | o Facilitate future changes |

Generally, redesign is dictated when the change is not feasible within the current structure of the system or in order to meet performance (efficiency) goals. Other compelling, but often overlooked reasons to redesign are the restructure of the code: to comply with standards; to incorporate more modern design and programming practices; and to facilitate future changes. A basic set of criteria for redesign may be:

o time and effort are available;

o feasibility of change requires it;

o performance requirement necessitates it;

o if more that 20% of the lines of code of a module are affected; and

o if a patch will cause the complexity of the system to exceed established standards.

## 9.2 Structured Versus Unstructured Coding Techniques

A significant percentage of the inventory of software being maintained in most organizations today was developed over eight years ago. This software may have been developed using unstructured languages and without the discipline of modern programming practices. In some situations, attempting to change this type of code using modern structured approaches is difficult. If the change is small and expediency takes priority, then no attempt to "structure" the code is necessary. If the change is significant, then redesign using more modern techniques are recommended. In this way, the overall structure of a software system can be gradually improved.

## 9.3 Regression Test Coverage

A key decision that must be made in all maintenance organizations is when a new version of a system should be placed into production. The question that must be answered is: has the system been tested sufficiently to provide the manager with confidence that the system will perform satisfactorily? Resource and time limitations typically prevent an organization from testing as much as they would like. The following guidelines are recommended: o If modifications to a system are localized, then regression testing should be performed against the subset of software effected. This subset can be determined by analyzing interfaces and determining that subset of software whose interfaces with the rest of the system are not affected by the change.

o If the modification permeates the system, regression testing should be performed. This is especially true where the change affects global data. The key to successful testing, however, is the proper selection of test data. See [MCCA77]. o During regression testing, criteria of complete testing are:

- execution without failure;
- 100% coverage of all decision paths; and
- 100% coverage of all interfaces.

## 9.4 Requirements Impact Analysis

Most change requests involve adding a new capability or modifying an existing capability. The user generally identifies more uses for the system or identifies more effective ways to use the system. The impact of a change should be assessed by the maintenance organization prior to making the change.

The manager should encourage the user to take a long range view of the system and plan changes to the system in a systematic way. One technique that has proven effective involves establishing and maintaining an itemized requirements data base which identifies each function and traces that function to the modules which implement it. This data base can then be used by managers to determine the feasibility and cost effectiveness of each change. The ability to estimate the probable resources required for each change has implications both in the resources and priority assigned to the requests. Use of the data base to analyze priority, impact, and long term goals with the user enables the manager to better plan for resource requirements.

## 9.5 Major Versus Minor Modification Assessments

The difference between a major modification and a minor one can vary significantly from one applications area to another. The set of criteria used to make the distinction of major and minor is installation dependent. What one manager may consider to be a large resource commitment, another may perceive as typical and acceptable.

Using a definite set of criteria will help the manager, as well as the user to understand the magnitude of each change. The manager must prepare information for each user request that will support any scheduling decisions. This information can often be prepared from historical data referencing past changes that were similar to the specific request.

# 10 SOFTWARE MAINTENANCE PROBLEMS TO AVOID

There are two key ingredients to successful ADP maintenance management. The first is to recognize problems that are unique to the ADP maintenance environment, and the second is to plan strategies and policies that incorporate maintenance technology, good management practices, and service goals of the organization.

## 10.1 Common Mistakes And Pitfalls To Avoid

From a management viewpoint, establishing set procedures (policies) and insuring that supervisors and personnel enforce those procedures is the best way to avoid problems and pitfalls. Many of these mistakes are a result of actions taken for the sake of expediency or from a lack of established procedures. Table 10.1-1 identifies common mistakes made in maintenance organizations, and describes the associated indicators or symptoms.

TABLE 10.1-1 COMMON MISTAKES AND PITFALLS

| MISTAKES/PITFALLS | SYMPTOMS/INDICATORS |
|---|---|
| o CM done manually by programmer | - Loss of configuration control |
| o No records kept (no change request form, problem report, or maintenance log) | - No resource tracking by task<br>- Inability to estimate task requirements<br>- Loss of configuration control |
| o Respond to problem reports and change requests only, | - Reactive maintenance |
| o No separation of maintenance function | - Transparent costs<br>- Lack of control/visibility |
| o Lack of organizational respect for maintenance, personnel management, use of junior/entry level programmers | - High turnover |
| o No quality assurance | - Cost of maintaining system gets higher each cycle<br>- Error rates after maintenance cycle are high |
| o No documentation updates | - Cost of maintaining system gets higher each cycle<br>- Loss of configuration cntl.<br>- Fixes become all patches to avoid ripple effect |
| o No redesign/all patch methodology | - System performance degrades<br>- Cost of maintaining system gets higher each cycle<br>- High turnover |
| o No formal regression testing | - Higher error rates after fielding<br>- customer dissatisfaction |

## 10.2 A Strategy For Implementing Survival Techniques

Implementation of survival techniques in an ADP maintenance environment requires the cooperation of the other ADP functional areas as well as the maintenance group. All parties involved must be convinced that the improvements are in their best interest and the overall goal is the improvement of the organizations's performance. Unless the implementation plan convinces all parties, provincial attitudes are likely to hinder any real improvements. Once some order is established, the maintenance manager can proceed to implement a plan to improve the overall performance of the maintenance organization. Consideration should be given to the following recommendations:

- If your maintenance organization is completely saturated and over-committed, has borrowed as many resources as possible from developing organizations, and management is considering a new application development which will dilute resources even further, look at off-the-shelf software. Many packages are available with maintenance support.

- If an application system has degraded in performance, maintenance is extremely difficult because of the complexity of the code, and documentation is outdated, consider forming an audit team. This audit team, consisting of at least one senior analyst, a programmer, perhaps a technical writer, and a user representative, could review all documentation, code, and records, and provide a documented recommendation of how to salvage the application system.

- If the workload is impossible to perform within required schedules, the manager should attempt to form a review group of users. The manager should chair this group and, at meetings, outline user requests, and schedule commitments, and available resources. Where conflicts exist, it will be the responsibility of this group to assign priorities and slip schedules.

- Develop a plan which provides a phased approach to the software maintenance process. This plan should introduce formal procedures for assigning responsibilities, and purchasing or developing tools. It should not only give direction to the staff, but help to improve the software products.

# 11 PERFORMANCE GOALS OF SOFTWARE MAINTENANCE MANAGEMENT

The overall goals of a maintenance group can be summarized as "keeping the system up and running" and "pleasing the users". This section describes some key measures of the performance of a maintenance organization which are listed as follows:

- o System availability and reliability
- o User satisfaction
- o Timely responses to change requests and problem reports
- o Productivity improvement

It is recommended that, as a minimum, these elements be measured when evaluating software maintenance performance goals.

## 11.1 System Availability And Reliability

Problem report frequency is a good measure of how well a maintenance organization is doing. This involves counting the problem reports that represent failure of the system to perform as specified. It does not include change requests which represent the user's desires for new capabilities or enhancements. The amount of downtime that results from a failure is an indicator of the severity of the problem. If the frequency of problem reports is steadily decreasing, then the maintenance organization is performing maintenance more effectively.

## 11.2 User Satisfaction

Although difficult to evaluate objectively, user satisfaction is the most important performance measure of a maintenance organization. The user's satisfaction with the maintenance organization is influenced, to a large extent, by an effective user interface. In some cases, the users lack in-depth knowledge of data processing. In other cases, the user may want to get involved in the technical issues of maintenance.

A technique that is effective in assessing user satisfaction is the use of periodic user surveys soliciting a critique of the maintenance organization's performance. Review of the results of this survey with the user and development of corrective action plans can be effective in improving user satisfaction.

## 11.3 Change Request/Problem Report Response Time

The time it takes to make a change to the system significantly influences the attitude about the maintenance organization. The attitude of upper level managers, as well as users, is influenced by the timeliness of completing each change request. Users want a response to their requests as soon as possible, and the upper level managers will focus on the cost factors involved in each alteration.

The change request and problem report forms can be used to capture statistics about response time. These statistics can be used to justify additional tools to increase productivity and to identify bottlenecks in procedures.

## 11.4 Productivity

Very few maintenance organizations use productivity measures, primarily because no standards or commonly accepted measures exist. However, using some measure of productivity is recommended because it aids in pinpointing areas where productivity gains can be realized. Some measures that are commonly used and that are relatively easy to compile include:

- number of problem reports corrected per time period;
- number of change requests completed per time period;
- number of lines of code (entire inventory of applications) maintained per maintenance personnel; and
- number of lines of code modified or added per maintenance personnel.

These productivity measures can be evaluated on an organization-wide basis or by application. Applications for which very low productivity figures are recorded may be candidates for redesign, documentation support, or support tools.

# 12 SUMMARY AND CONCLUSIONS

Software maintenance effort is impacted by the fact that a large percentage of the current inventory of applications being maintained was not designed to accommodate change. A significant number of systems are poorly implemented and poorly documented due to the lack of discipline, the lack of uniformity in development approaches, and schedule and budget pressures. Thus, the use of state-of-the-art development methodologies will not guarantee a software system in which all future changes are anticipated.

Our findings indicate that there is a need to:

- o adopt a software engineering approach for software maintenance which takes into consideration software problems, personnel involved, and user and management constraints, (i.e. schedules, cost, and system capability);

- o develop management policies and procedures which incorporate maintenance requirements. This implies establishing a maintenance philosophy during the early stages of software development, as well as a maintenance technology which defines software maintenance requirements, activities, techniques, and tools. The overriding objective of this technology should be understandable, maintainable software which is easy to revise and validate over the software life cycle; and

- o adopt enforceable controls. Any clarification or redefinition of maintenance may provide scant savings if not accompanied by controls which can be made to work within the organization. Such controls will enable improved error tracking and also help to insure that software change requests:

  -are within the scope of maintenance;
  -do not adversly affect software performance;
  -are incorporated using a modular approach;
  -are adequately tested; and
  -are properly documented.

It must be recognized that the problems faced in most software maintenance environments are twofold: improving maintainability and convincing management that the greatest gain will be realized only when maintainability is engineered into the software products. The goal of maintainability must be established during the requirements, design, and development stages and built into the software using standards, and proven design and development techniques. It is also essential to involve the managers, maintainers, and users during each of these stages. The early introduction of maintenance concepts,

coupled with the use of more effective techniques for maintenance, will lead to an integrated approach to software maintenance over the lifecycle. The overall effect will be increased maintainability and increased cost effectiveness of the ADP organization in general. Therefore, maintainability must be a goal of development.

The following three tables provide general guidance on practices that, when followed, will result in more maintainable software. Table 12.1-1 provides a list of general practices found to be effective for improving software maintainability. Table 12.1-2 identifies factors and describes the activities necessary to ensure that maintainability is engineered into the software products, while Table 12.1-3 provides a checklist of attributes that can be used as guidance to help programmers improve the quality and maintainability of the software. This guidance ranges from conceptual (anticipating the need to change in design and implementation), to incorporating state-of-the-art technology (using modern programming practices), to managerial (transition development tools to maintenance environment), to practical (have maintainer be involved in development). If these concepts are consistently enforced, the resultant software product will be easier to understand, correct, modify, and maintain.

TABLE 12.1-1 IMPROVING THE MAINTAINABILITY OF SOFTWARE

- Design and Implement for Maintainability/ Enhancement (Top down)
- Use High Order languages (4th Generation languages if possible)
- Use Report Generators, Data Base Management Systems, System Utilities, Support Software
- Use Modern Programming Practices/Techniques
- Document Design and Implementation
- Trace Requirements to Implementation
- Save Test Data
- Use Error Checking Techniques
- Design for Human Engineering
- Move Development Tools to Maintenance Environment

Table 12.1-2 MAINTENANCE CONSIDERATIONS DURING DEVELOPMENT

| Phases | Activity |
|---|---|
| Requirements Specification | o Identify maintainability as a primary goal of development.<br>o Identify maintenance Facility/ Resource requirements.<br>o Document requirements. |
| Design Phase | o Establish design standards and conventions (maintainability supporting standards.<br>o Audit design for compliance with standards.<br>o Document design.<br>o Document data base design.<br>o Trace requirements to design. |
| Coding Phase | o Establish coding standards and conventions (maintainability supporting standards).<br>o Audit code for compliance with standards).<br>o Trace design to code.<br>o Document code.<br>o Comment code.<br>o Document a maintenance manual.<br>o Transition code and debug tools to maintenance. |
| Test Phase | o Document all errors.<br>o Assess maintainability during error correction.<br>o Maintain code and documentation during test/error correction activities.<br>o Establish test data base<br>o Transfer test tools to maintenance environment |

TABLE 12.1-3 ATTRIBUTES OF MAINTAINABLE SOFTWARE [MCCA77]

| | |
|---|---|
| Maintainability | - the effort required to locate and fix an error in operational program or the effort required to modify an operational program (Flexibility). |
| Consistency | - those attributes of software that provide uniform design and implementation techniques and notation. |
| Simplicity | - those attributes of the software that provide implementation of functions in the most understandable manner (usually avoidance of practices which increase complexity). |
| Conciseness | - those attributes of the software that provide for implementation of a function with a minimum amount of code. |
| Modularity | - those attributes of the software that provide a structure of highly independent modules. |
| Self-Descriptiveness | - those attributes of the software that provide explanation of the implementation of a function. |
| Generality | - those attributes of the software that provide breadth to the functions performed. |
| Expandability | - those attributes of the software that provide for expansion of data storage requirements or computational functions. |

## 13 REFERENCES


[AFTE80] AFTEC/TEBC, "Software OT&E Guidelines, Vol. III: Software Maintainability Evaluator's Handbook." NTIS ADA 104328, Apr. 1980.

[AFLO77] Alford, M., "A Requirements Engineering Methodology for Real-Time Processing Requirements," IEEE Transactions on Software Engineering, Vol 3, 1977.

[BOEH76] Boehm, B., "Software Engineering," IEEE Transactions on Computers, Vol. C-25, No. 12, Dec. 1976.

[CIRA71] (IRAD), "A Study of Fundamental Factors Underlying Software Maintenance Problems: Final Report" ESD-TR-72-121, Dec. 1971.

[COOK80] Cook, J., "An Appraisal of Selected Cost/ Resource Estimation Models for Software Systems," NASA X-582-81-1, Dec. 1980.

[DERO79] DeRoze, B., Nyman, T., "The Software Lifecycle - A Management and Technological Challenge in the Department of Defense," IEEE Transactions on Software Engineering, Vol 4, 1979.

[DONA80] Donahoo, J. et al, "A Review of Software Maintenance Technology," RADC-TR-80-13, Feb. 1980.

[GAO80] GAO, "Wider Use of Better Computer Software Technology Can Improve Control and Reduce Costs," FGMSD-80-38, Apr. 1980.

[GAO81a] GAO, "Government-Wide Guidelines and Management Assistance Center Needed to Improve ADP Systems Development," AFMD-81-20, Feb. 1981.

[GAO81b] GAO, "Federal Agencies' Maintenance of Computer Programs: Expensive and Undermanaged," Report to Congress AFMD-81-25, Feb. 1981.

[GELP79] Gelperin, D., "Testing Maintainability," ACM SIGSOFT Software Engineering Notes, Vol. 4/No. 2, Apr. 1979.

[GILB79] Gilb, T., "A Comment on 'The Definitions of Maintainability'," ACM SIGSOFT Software Engineering Notes, Vol. 4,/No. 3, Jul. 1979.

[HALS77] Halstead, M., Software Science. Excelier Press, 1977.

[HERN81] Herndon, M., "Development of the Requirements Management Methodology," SAI Report LJF-81-0071/OFS OSS, Jan. 1980.

[HOUG82] Houghton, R., "Software Development Tools," NBS Special Pub 500-88, Mar. 1982.

[LIEN76] Leintz, B. and Swanson, E.B., "Characteristics of Application Software Maintenance," NTIS ADA 024085, Dec. 1976.

[MART82] Martin, R., Osborne W., "Guidance on Software Maintenance," NBS Special Publications 500-106, December 1983.

[MART83] Martin, J., McClure, C., Software Maintenance: The Problem and Its Solution, Prentice-Hall, Inc., New Jersey, 1983.

[MCCA76] McCabe, T., "Complexity Measure," Proc 2nd International Conference on Software Engineering, 1976.

[MCCA77] McCall, J., et al, "Factors in Software Quality," RADC-TR-77-369, Nov. 1977.

[MCCA83] McCall, J., Herndon, M., "Software Maintenance Survey," NBS Contract NB82SBCA1650, June 1983.

[MYER75] Myers, G., Reliable Software Through Composite Design, Petrocelli/Charter, 1975.

[MYER77] Myers, G., Software Reliability: Principles and Practices, John Wiley & Sons, 1977.

[PARI82] Parikh, G., Techniques of Program and System Maintenance, Winthrop Publisher, Inc., 1982.

[PARI83] Parkh, G., Zvegintzov, N., "Tutorial on Software Maintenance," IEEE Computer Society, Catalog No. EH0201-4, 1983.

[RADC82] RADC DACS Newsletter, June 1982.

[STAN77] Stanfield, JR., "Software Acquisition Management Guidebook: Software Maintenance:" AD A052040, Oct. 1977.

[SWAN76] Swanson, E.B., "The Dimensions of Maintenance," Proc Second Conference on Software Engineering, 1976.

[TEIK77] Teichroew, D., Hershey, E., "PSL/PSA: A Computer-aided Technique for Structured Documentation and Analysis of Information Processing Systems," IEEE Tranactions on Software Engineering, vol 3, 1977.

[THIB81] Thibodeau, "An Evaluation of Software Cost Estimation Models," RADC TR, Feb. 1981.

[WHIT77] Whitmore, D.C., "Computer Program Maintenance One of the Software Acquisition Engineering Guidebook Series," NTIS ADA083209, Dec. 1977.

NBS-114A (REV. 2-8C)

| U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET** (See instructions) | 1. PUBLICATION OR REPORT NO. NBS/SP-500/129 | 2. Performing Organ. Report No. | 3. Publication Date October 1985 |
|---|---|---|---|

4. TITLE AND SUBTITLE

Computer Science and Technology:
Software Maintenance Management

5. AUTHOR(S)

James A. McCall, Mary A. Herndon, Wilma M. Osborne

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

**NATIONAL BUREAU OF STANDARDS**
**DEPARTMENT OF COMMERCE**
~~WASHINGTON, D.C. 20234~~
Gaithersburg, MD 20899

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: 85-600596

☐ 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)

Software maintenance is a critical support function within most large organizations today. In spite of its importance to the organization, software maintenance has been ignored as a significant management concern and as a fertile area for technical improvement. This report presents an overview of the various aspects of software maintenance, and provides an indepth analysis of the associated problems, giving particular attention to the most pressing ones. It identifies tools, techniques, and procedures which aid in reducing these problems. This report also provides detailed guidance for managing software maintenance as a separate organizational entity. It also provides assistance needed to develop and employ improved maintenance practices and procedures, that result in reduced software costs and which help to insure that quality software is developed for and by the Federal ADP community.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

cost control measures; decision aids, lifecycle management plan; performance criteria; regression testing; software maintenance; software quality; software techniques; software tools; test plans; software quality

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

65

15. Price

USCOMM-DC 6043-P80

## ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SCIENCE & TECHNOLOGY

Superintendent of Documents,
Government Printing Office,
Washington, DC 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

## Periodical

**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. Issued six times a year.

## 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 Journal of Physical and Chemical Reference Data (JPCRD) is published quarterly for NBS by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements are 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 Service, 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 Service, Springfield, VA 22161, in paper copy or microfiche form.

**U.S. Department of Commerce**
National Bureau of Standards
Gaithersburg, MD 20899

Official Business
Penalty for Private Use $300