(12) LEVEL II

RADC-TR-80-109, Vol I (of two)
Final Technical Report
April 1980

ADA086985

SOFTWARE QUALITY METRICS ENHANCEMENTS

General Electric Company

James A. McCall
Mike T. Matsumoto

1d2
A086986

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC
S ELECTE D
JUL 22 1980
B

DDC FILE COPY

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE NY 13441

US ARMY INSTITUTE FOR RESEARCH IN
MANAGEMENT INFORMATION AND COMPUTER SCIENCES
ATLANTA GA 30332

80 7 21 01

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

RADC-TR-80-109, Volume I (of two) has been reviewed and is approved for publication.

APPROVED: Joseph P. Cavano

JOSEPH P. CAVANO
Project Engineer

APPROVED: Wendall C Bauman

WENDALL C. BAUMAN, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

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

Do not return this copy. Retain or destroy.

(19) TR-80-109-VOL-1

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (18) RADC-TR-80-109, Vol I (of two)	2. GOVT ACCESSION NO. AD-A086 985	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) SOFTWARE QUALITY METRICS ENHANCEMENTS. Volume I.		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report June 1978 - July 1979
		6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) (10) James A. McCall (7.) Mike Matsumoto		8. CONTRACT OR GRANT NUMBER(s) (15) F30602-78-C-0216
9. PERFORMING ORGANIZATION NAME AND ADDRESS General Electric Company — 149 450 Information Systems Programs 450 Persian Drive, Sunnyvale CA 94086		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 63728F (16) 25280202 (17) 02
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIS) Griffiss AFB NY 13441		12. REPORT DATE April 1980
		13. NUMBER OF PAGES 182
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same (12) 187		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

(11) Apr 80

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

Same (9) Final rept. Jun 78-Jul 79

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Joseph P. Cavano (ISIS) 315 330-4325

USACSC Project Engineer: Daniel E. Hocking (AIRMICS) 404 894-3111

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Quality
Quality Metrics
Software Measurement

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

Software metrics (or measurements) which predict software quality have been refined and enhanced. Metrics were classified as anomaly-detecting metrics which identify deficiencies in documentation or source code, predictive metrics which measure the logic of the design and implementation, and acceptance metrics which are applied to the end product to assess compliance with requirements. over (Cont'd)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

149 450

A Software Quality Measurement Manual was produced which contained procedures and guidelines for assisting software system developers in setting quality goals, applying metrics and making quality assessments.

PREFACE

This document is the final report (CDRL A003) for the Metrics Enhancement Study, contract number F30602-78-C-0216. The contract was performed in support of the Air Force Systems Command Rome Air Development Center and the U.S. Army Computer Systems Command Army Institute for Research in Management Information and Computer Sciences.

The report was prepared by Mike Matsumoto and Jim McCall of the Sunnyvale Operations, Information Systems Programs, General Electric Company. The Program Manager was Gene Walters. Significant contributions were made by Sue Ehnert and Bob Hassell.

Technical guidance was provided by Joe Cavano, RADC Project Engineer and Dan Hocking, USACSC Technical Monitor.

The report consists of two volumes as follows:

Volume I Software Quality Metrics Enhancements
Volume II Software Quality Measurement Manual

Volume I provides a description of the research activities performed during the contract. Volume II provides a manual describing how to apply the metrics oriented toward quality assurance personnel.

ACCESSION for
NTIS White Section ☑
DDC Buff Section □
UNANNOUNCED □
JUSTIFICATION ________
BY ________
DISTRIBUTION/AVAILABILITY CODES
Dist. AVAIL. and/or SPECIAL
A

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF TABLES (Continued)

EVALUATION

The purpose of this research was to refine and enhance the software quality measurement process that was originally documented in RADC TR-77-369. The work covered by this effort is contained in two volumes. The first volume includes extensions to the concepts of software quality measurement, analysis of metric applications and validation of metrics for the quality factors portability and maintainability. Appendix D of Volume I documents all the changes that have been made to the software quality metrics based on the experiences of this research study.

The second volume of this report, A Software Quality Measurement Manual, is oriented toward the quality assurance process and identifies how to set quality goals, how and when to apply software metrics and how to make a quality assessment.

This effort was initiated in response to RADC TPO5, Software Cost Reduction, in the area of Quality Measurement. The effort was co-sponsored by the US Army Computer Systems Command, Army Institute for Research in Management Information and Computer Science.

This work was significant because it verified that metrics originally developed for Air Force Command and Control applications were also applicable to Army Management Information System applications. It is anticipated that the metrics will be applicable to other environments as well. In addition, the normalization functions developed for portability increases the degree of confidence that can be placed on the quality measurements. The fruitful results of this study should prove to be of great aid in complementing existing software quality assurance techniques by providing quantitative quality descriptions of software during the development cycle itself.

Joseph P. Cavano

JOSEPH P. CAVANO
Project Engineer

SECTION 1
INTRODUCTION

1.1 REPORT OVERVIEW

The Metrics Enhancements task, contract no F30602-78-C-0216, was conducted in support of the Air Force Systems Command Rome Air Development Center's (RADC) and U. S. Army Computer Systems Command's (USACSC) missions to investigate, sponsor, and develop techniques which enhance the development of high quality software. The inputs for this effort, the background of previous related work, the task objectives, and the scope of this final report are described in this section.

1.1.1 OVERALL TASK OBJECTIVES

The major goal of this research effort was to further test the feasibility and validity of the concept of software quality metrics established during the Factors in Software Quality Contract, Contract number F30602-76-C-0417. In order to accomplish this goal, the following tasks were performed (figure 1.1-1).

1.1.2 TASK 1: ANALYSIS OF SOFTWARE QUALITY METRICS

The initial task involved analysis of the set of metrics established in RADC TR-77-369 for applicability to a Management Information System (MIS) software production environment. The analysis consisted of an evaluation of each metric with respect to MIS applications and the COBOL programming language, and was based on lessons learned from the previous effort. In performing the analysis, the products produced during typical software developments were identified and the metrics related to those products here assessed for applicability. In order to provide a complete evaluation of the applicability of the concept of software quality metrics to an MIS environment, an evaluation of the differences in quality requirements between Command and Control (C^2) applications and MIS applications was made. The results of this task are in section 2 of this report.

TASK RELATIONSHIPS AND PRODUCTS

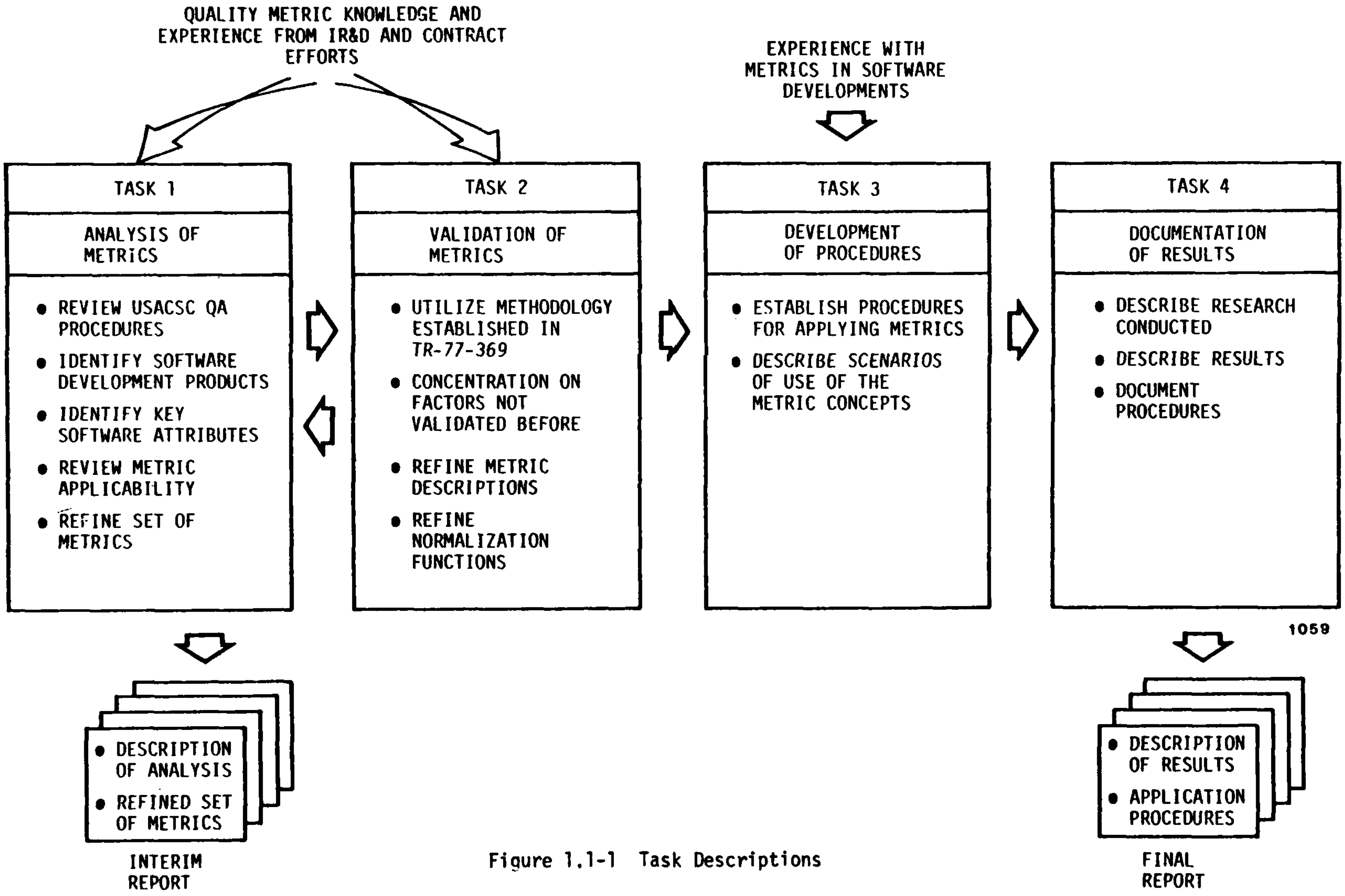

QUALITY METRIC KNOWLEDGE AND EXPERIENCE FROM IR&D AND CONTRACT EFFORTS
EXPERIENCE WITH METRICS IN SOFTWARE DEVELOPMENTS
TASK 1
ANALYSIS OF METRICS
● REVIEW USACSC QA PROCEDURES
● IDENTIFY SOFTWARE DEVELOPMENT PRODUCTS
● IDENTIFY KEY SOFTWARE ATTRIBUTES
● REVIEW METRIC APPLICABILITY
● REFINE SET OF METRICS
TASK 2
VALIDATION OF METRICS
● UTILIZE METHODOLOGY ESTABLISHED IN TR-77-369
● CONCENTRATION ON FACTORS NOT VALIDATED BEFORE
● REFINE METRIC DESCRIPTIONS
● REFINE NORMALIZATION FUNCTIONS
TASK 3
DEVELOPMENT OF PROCEDURES
● ESTABLISH PROCEDURES FOR APPLYING METRICS
● DESCRIBE SCENARIOS OF USE OF THE METRIC CONCEPTS
TASK 4
DOCUMENTATION OF RESULTS
● DESCRIBE RESEARCH CONDUCTED
● DESCRIBE RESULTS
● DOCUMENT PROCEDURES
1059
● DESCRIPTION OF ANALYSIS
● REFINED SET OF METRICS
INTERIM REPORT
● DESCRIPTION OF RESULTS
● APPLICATION PROCEDURES
FINAL REPORT

Figure 1.1-1 Task Descriptions

1.1.3 TASK 2: VALIDATION OF THE METRICS

The methodology established in RADC TR-77-369 was utilized to apply the metrics to the USACSC Modernized Army Research and Development Information System (MARDIS) development data base as a typical MIS application and to th General Electric/Integrated Software Development System (GE/ISDS) maintenance data base (see Appendix A for brief descriptions of these data bases) as a typical software support system. In this context the data bases refer to all of the documents, flowcharts, and source code associated with the development effort for these systems. The establishment and validation of normalization functions (the mathematical relationships which relate metrics to ratings of the various quality factors) for factors which were not validated previously were given most attention. Based on the application of the metrics and the validation process, further refinements to the metrics were made. The results of this task are in section 3 of this report.

1.1.4 TASK 3: DEVELOPMENT OF PROCEDURES

Based on the experience gained during the validation, detailed guidelines and procedures were developed for applying the metrics. These guidelines and procedures are oriented toward application by Quality Assurance (QA) personnel and interpretation of the results by program managers. These results may be seen in the Software Quality Measurement Manual, an attachment to this report.

1.1.6 SCOPE OF FINAL REPORT

This final report represents satisfaction of CDRL A003 of the Metrics Enhancements contract. It describes the technical effort and results of the previously mentioned tasks. The report includes a description (section 3) of the following:

- Identification of difference in quality requirements between typical C^2 and MIS applications.
- Description of documentation produced in Air Force and Army software developments

- Review and refinement of software quality metrics in light of MIS applications
- Extensions of the concept of software quality metrics.
- Results of a validation of metrics using an Army MIS system and a software support system.

1.2 THE CONCERN FOR SOFTWARE QUALITY

A brief review of the evolution of major areas of concern in the software engineering field over the last decade and the direction of research pursued as a result of that concern provides support and a historical perspective of the current emphasis on software quality.

The genesis of software engineering and structured programming, in terms of community-wide recognition and publication in the literature, is usually traced to the 1968 NATO Conference on Software Engineering and Edsgar Dijkstra. At that conference, Dijkstra noted how encouraging it was to see the extremely well-qualified attendees admitting that problems existed in the development of software. He felt that the first step towards solving the problem was recognizing the problem [DIJE69]. The direction of the research community during this time period was toward solidifying the concepts of software engineering and structured programming, and identifying the problem areas of software development [BOEB73], [STR74].

The Symposium on the High Cost of Software in 1973 sponsored by the three Services can be viewed as another key event in the expression of major concerns in the software community. During this conference, the problem of the high cost of software and the increasing proportion of system development costs attributed to software were the focal points [PRO73]. The direction of research in the ensuing years emphasized improving the productivity of programmers. This direction manifested itself in the development of tools and aids to assist in the very labor-intensive process of software development. While many significant results have been achieved in this area, the attack was on a symptom (high costs) rather than a problem, and in an area which provided relatively low leverage. Programming has been shown to only account for approximately 20% of the total software developments cost (Design 40%, Test 40%).

The International Conference on Reliable Software in 1975 was the forum of a sharper focus of concern [PRO75]. Here the theme was software reliability and the concern for the very critical problem of unreliable software. During this time period, the research community responded with error data collection efforts, error classification studies, reliability modeling studies in an attempt to bound and define the problem. Many of these efforts are currently beginning to show results.

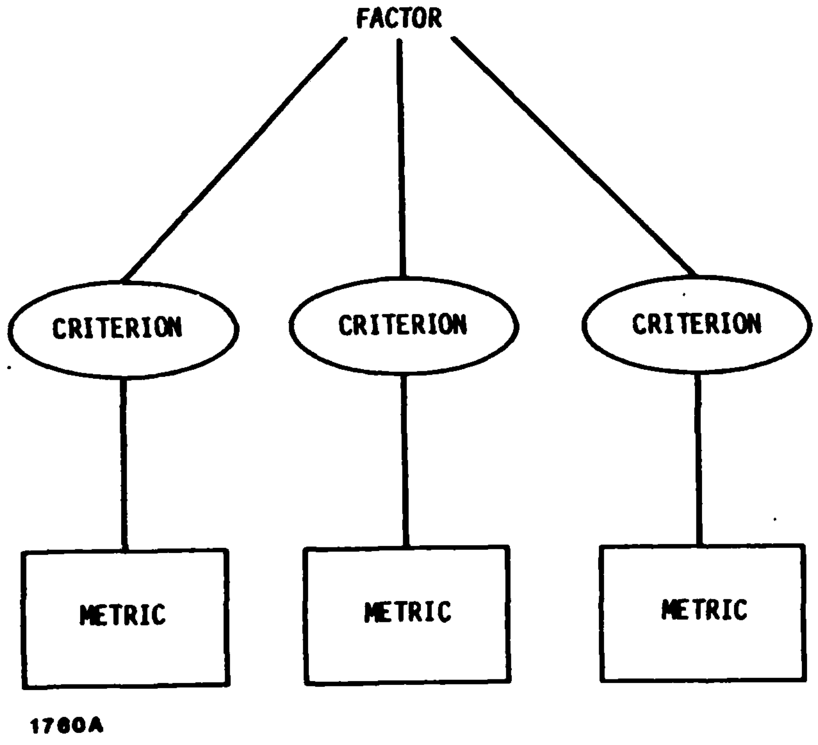
Each step mentioned above has provided some progress. Products of this research during the past decade have had significant impact on the way we develop software. Where have we evolved since that point? What are the major research concerns today? 1979 finds us looking at a larger problem - the quality of our software systems. The quality of software, a part of which is reliability and a measure of which is cost, has become a major concern because it has been recognized that software costs do not end at delivery. The concern now is for life cycle costs, total costs, and user satisfaction throughout the life of the system, not just at delivery.

Life cycle management and life cycle costs have become the major concerns. The leverage in this approach can be seen in statistics that identify 60 to 80% of life cycle costs as being post-delivery costs. Thus, a major direction in research today is software quality with a perspective on software from a life cycle viewpoint.

The software quality metrics concepts which are the subject of this report provide a mechanism for addressing software life cycle considerations.

1.3 FACTORS IN SOFTWARE QUALITY TASK

This effort is a continuation and extension of research in software quality metrics sponsored by RADC, contract no. F30602-76-C-0417. In that previous effort a framework for addressing the subject of software quality and its measurement was established. This framewo.·k, shown in figure 1.3-1, has three levels. At the highest level, the level at which management



- MANAGEMENT-ORIENTED VIEW OF PRODUCT QUALITY

- SOFTWARE-ORIENTED ATTRIBUTES WHICH PROVIDE QUALITY

- QUANTITATIVE MEASURES OF THOSE ATTRIBUTES

Figure 1.3-1 Software Quality Framework

and users interface with the framework, are management-oriented terms identifying the major aspects of software quality. These terms, called quality factors, are shown with their definitions in table 1.3-1. At the next level, sets of attributes of the software which contribute to the characteristics represented by the quality factors are identified. Then, at the lowest level, are quantitative measures, metrics, of those attributes. All of the attributes, or criteria, are shown in figure 1.3-3 as they relate to the factors. The metrics are discussed later in section 2. This framework and the definitions represent an hierarchical definition of software quality, the hierarchy involving different levels of detail and different orientations or viewpoints of software quality.

Another product of the previous effort was the establishment of a methodology for the validation of the metrics. This methodology consists of the following steps:

(1) Application of the metrics to the products generated during a software development. The products include documentation such as requirements specifications, design specifications, test plans, users manuals, as well as the source code.

(2) Utilizing development and operational historical data, rating of the software by quality factor can be derived. Using these ratings as dependent variables and the values obtained from the application of the metrics as independent variables, a multivariate regression analysis can be performed. The resulting equation, a normalization function, provides a mathematical relationship between the metrics and the quality factors.

(3) Validation of these normalization functions was performed by plotting the same data (ratings and metric values) for other systems or modules and deciding whether they fall within a 90 percent confidence interval. The 90 percent confidence interval was chosen as the validation criteria because it provides sufficient precision for analysis to be done using the normalization functions.

Table 1.3-1 Software Quality Factors

		FACTORS	DEFINITIONS
LIFE CYCLE STAGES	INITIAL PRODUCT OPERATION	CORRECTNESS	Extent to which a program satisfies its specifications and fulfills the user's mission objectives.
		RELIABILITY	Extent to which a program can be expected to perform its intended function with required precision.
		EFFICIENCY	The amount of computing resources and code required by a program to perform a function.
		INTEGRITY	Extent to which access to software or data by unauthorized persons can be controlled.
		USABILITY	Effort required to learn, operate, prepare input, and interpret output of a program.
	PRODUCT REVISION	MAINTAINABILITY	Effort required to locate and fix an error in an operational program.
		TESTABILITY	Effort required to test a program to insure it performs its intended function.
		FLEXIBILITY	Effort required to modify an operational program.
	PRODUCT TRANSITION	PORTABILITY	Effort required to transfer a program from one hardware configuration and/or software system environment to another.
		REUSABILITY	Extent to which a program can be used in other applications - related to the packaging and scope of the functions that programs perform.
		INTEROPERABILITY	Effort required to couple one system with another.

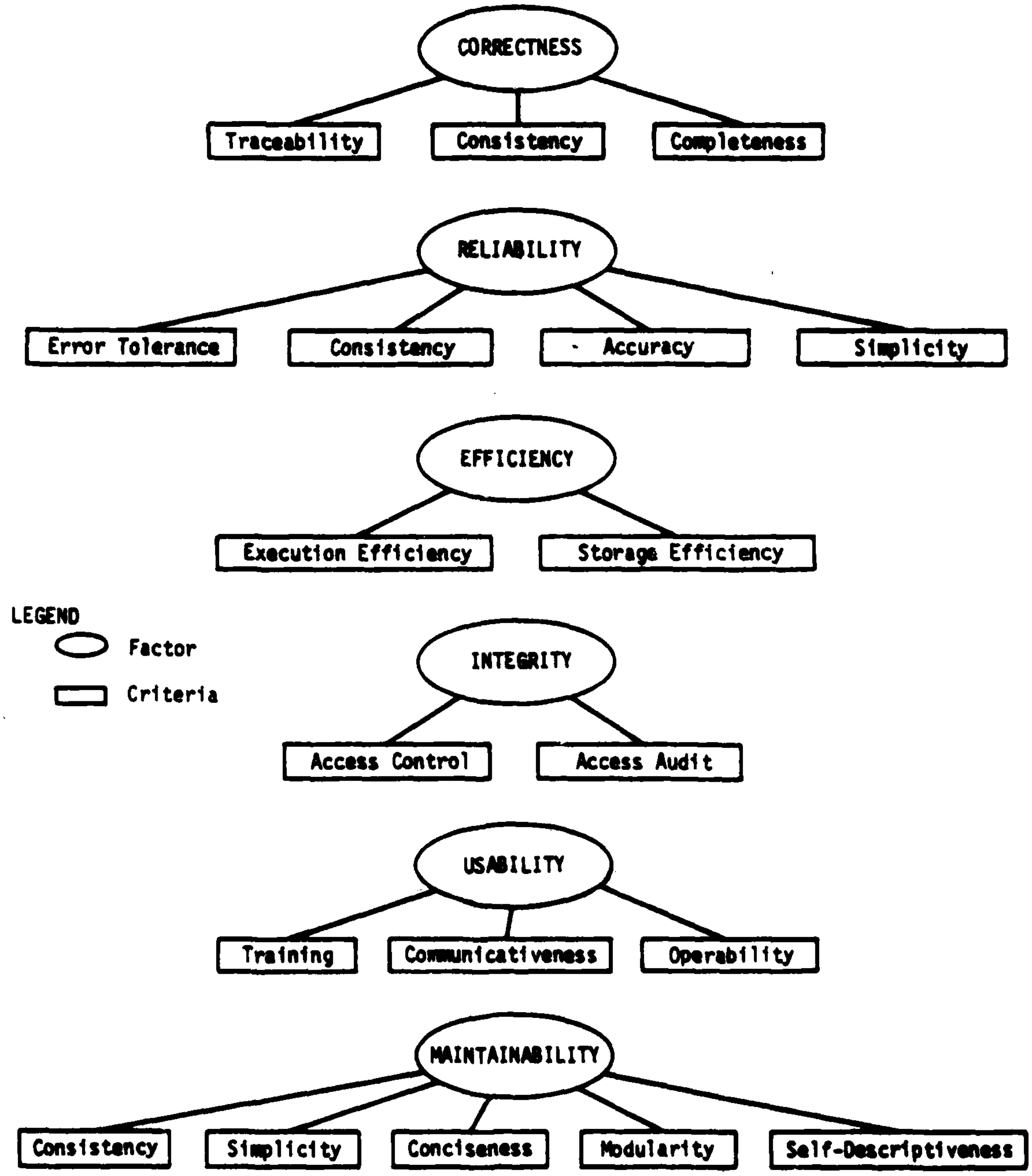


Figure 1.3-3 Relationship of Criteria to Software Quality Factors

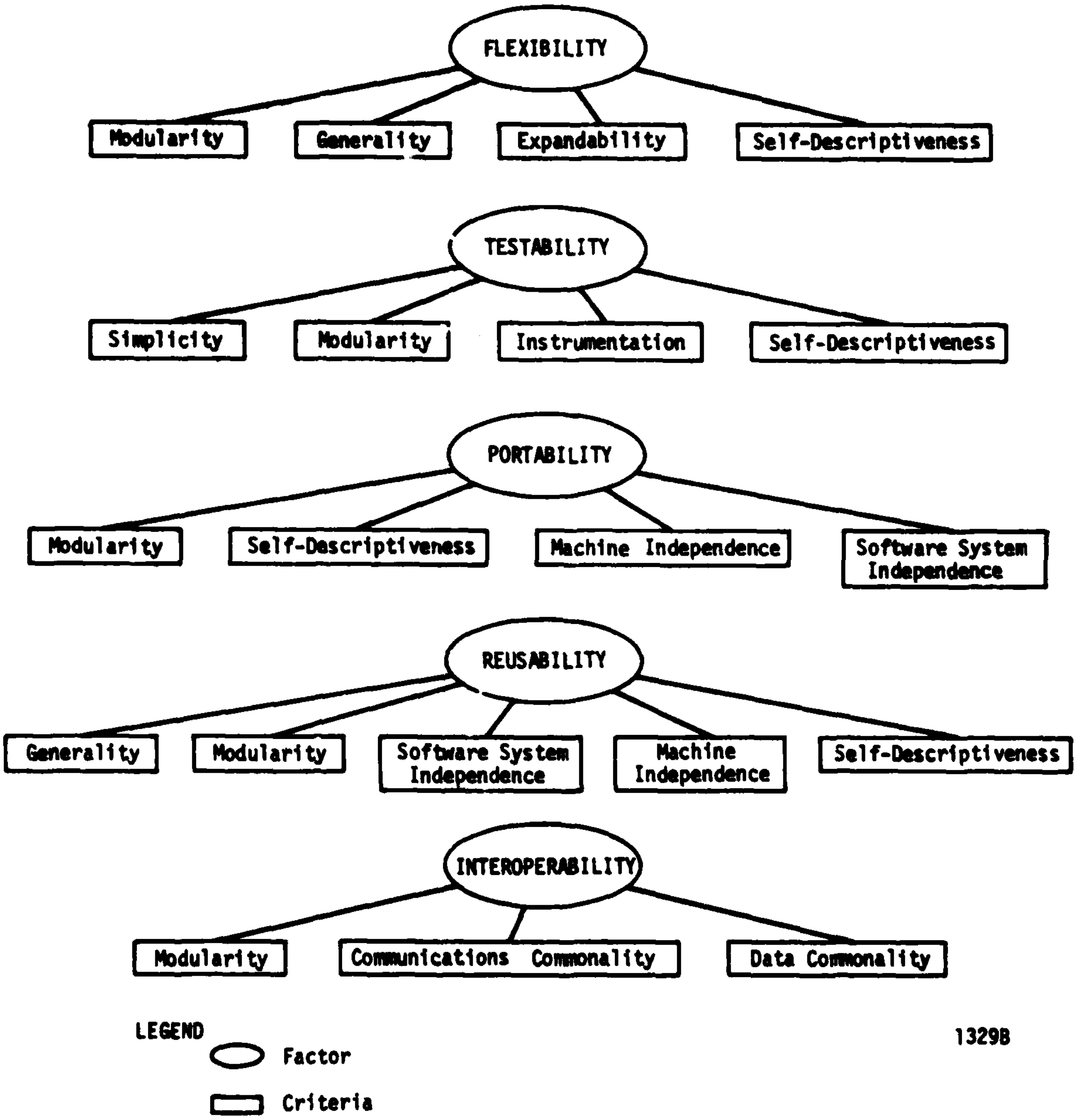


Figure 1.3-3 Relationship of Criteria to Software Quality Factors (continued)

This methodology is illustrated in figure 1.3-4.

Two other results of the previous effort were the identification of automated support tools which could be utilized to apply the metrics and the documentation of a preliminary handbook. The handbook was oriented toward an acquisition manager and described our concept of software quality and three approaches (each one more detailed than the other) for specifying and measuring software quality. These results are the assumed starting point for this current effort.

This previous research indicated that the concept of measurable software quality was a pragmatic approach to improved software. However, the experience with the metrics was limited to the command and control environment of the Air Force. Major differences exist in the factors essential to software quality between a Command and Control (C^2) System and a Management Information System (MIS). In the former, the emphasis may be on reliability and efficiency; in the latter, the emphasis may be on portability and maintainability. These differences require that additional experience be gained in the use of metrics in other environments and with different applications.

An additional deficiency exhibited by the previous research in software quality that a broad based confidence in all of the quality factors/metrics had not been achieved. The two Air Force software systems used in the previous study had not experienced some of the activities necessary to validate metrics related to certain quality factors. As an example, neither system had been moved to another environment and therefore none of the metrics associated with the quality factor, portability, were able to be validated. The USACSC's recent experience in transferring systems to other hardware environments and investigation into the portability of software provided an excellent basis for validation and refinement of the metrics related to portability.

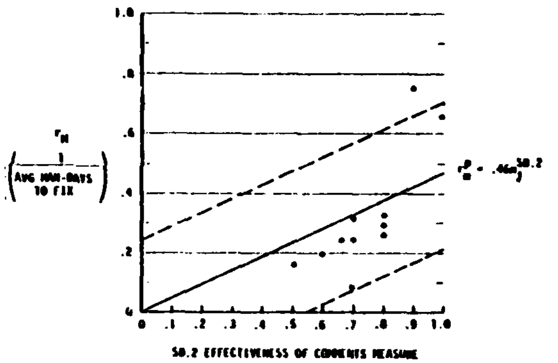
The USACSC Modernized Army Research and Development Information System (MARDIS) data base affords an excellent testbed for the application of software metrics in an MIS environment. In addition to utilizing the MARDIS system

FACTOR(S): FLEXIBILITY, MAINTAINABILITY, TESTABILITY, PORTABILITY, REUSABILITY (CONTINUED)

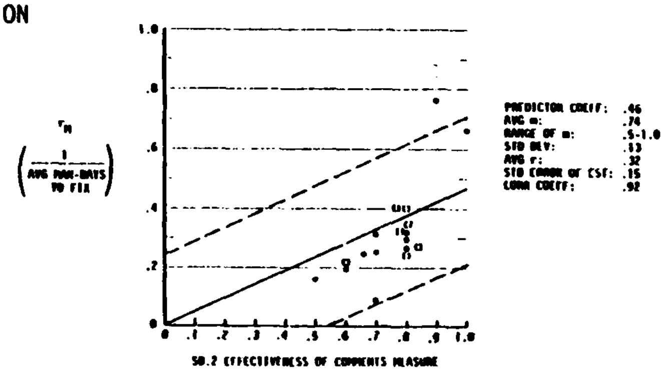
CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR 0	VALUE	YES/NO 1 OR 0	VALUE	YES/NO 1 OR 0	VALUE
EFFECTIVENESS OF COMMENTS (p)	SD. 2 EFFECTIVENESS OF COMMENTS MEASURE						
	(1) Modules have standard formated prologue comments which describe: - Module name/version number - Author - Date - Purpose - Inputs - Outputs - Function - Assumptions - Limitations and restrictions - Accuracy requirements - Error recovery procedures - References $1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}$						3111
	(2) Comments set off from code in uniform manner $1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}$						3111
	(3) All transfers of control & destinations commented $1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}$						3111
	(4) All machine dependent code commented $1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}$						3111
	(5) All non-standard HOL statements commented $1 - \frac{\text{\# modules violate rule}}{\text{total \# module}}$						3111
	(6) Attributes of all declared variables commented $1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}$						3111
	(7) Comments do not just repeat operation described in language $1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}$						3111

CRIS FORM 140 (13 JAN 77)

STEP 1: APPLICATION OF METRICS



STEP 2: ESTABLISH NORMALIZATION FUNCTIONS



STEP 3: VALIDATION

Figure 1.3-4 Application and Validation of Metrics

as an example of an MIS system, a state-of-the-art software support system (GE/ISDS) developed at GE/Sunnyvale was also analyzed. This system was transferred to a number of GE locations and data was collected on the effort required to accomplish the transfer. Utilization of this data provided experience in applying the metrics to a software support system environment and allowed additional validation of metrics related to portability and maintainability.

1.4 SUMMARY OF FINDINGS

As a result of the metrics Enhancement study the following conclusions can be stated (reference is made to paragraphs in this report providing supporting data):

1) The framework established in RADC-TR-77-369 is applicable to other environments and provides a useful life cycle management viewpoint to software system requirements specification (2.2)
2) The metrics established in RADC-TR-77-369 have now been applied to two JOVIAL command and control systems ($\sim$ 40,000 lines of code), to a financial management information system ($\sim$ 54,000 lines of code), and to a FORTRAN software support system ($\sim$ 20,000 lines of code) (3.1)
3) The metrics and the metric worksheets created for their manual application provide a quantitative evaluation tool for quality assurance personnel (2.6.3, 3.3.1 Vol. II).
4) Sensitivity Analyses based on the quantitative measures provide an immediately applicable quality assurance technique (3.3.2)
5) The concept of Software Quality Metrics is supported by statistical analyses although because of limited samples, further research is needed before a high degree of confidence can be placed on the mathematical relationships established to date (3.3.3)
6) Because techniques derived during the software metric research seem to have potential as quality assurance tools, a Software Quality Measurement Manual has been developed to provide guidance for establishing quality goals for a software development and measuring the programs toward those goal during the development (Vol II).

SECTION 2

EXTENSIONS TO THE CONCEPTS OF SOFTWARE QUALITY METRICS

2.1 INTRODUCTION

Recent literature in the field of Software Engineering has placed increased importance on Life Cycle Management. The realization that software has become the most expensive factor in computer systems has caused the emphasis on the efficient management of the software life cycle.

Throughout this report we will discuss the concept of software quality in the context of the MIS environment and the C^2 environment. In terms of extending these concepts to a full program for the management of the quality of software, however, we may take different perspectives on the nature of software systems. These perspectives are normative and based on the goals of the organizations which develop software; they should not be construed as absolute pronouncements on the nature of software.

2.2 PRODUCTS AND SERVICES

Software developers have a tendency to view programs as static, finished products once they have gone into the operational phase of the life cycle. Most of us, however, are well aware of the fact that this is a chimera, that in fact, software has a complete life cycle and goes through maintenance and enhancement phases before its final obsolesence.

Users of software systems have a different outlook. Software usually performs a service for the user and, developers then become the providers of the system which provides the needed service.

In the C^2 environment, where software systems generally are part of larger (embedded) systems, the product orientation is a convenient one which allows programmers, engineers, and users to manage the life cycle with respect to the product in which the software is embedded.

Of particular importance in the MIS environment, however, is the fact that the development staff almost invariably acts as a support unit to the primary function of the organization. For the most part, the programs are not in themselves the products which the user organization ultimately produces. In this sense, the developer's staff performs a service for the rest of the organization, and so the task of the developer's staff is one of providing that service to the functional (user) components of the organization. User components are not in themselves interested in the technical aspects of programming, or even in that of systems analysis, but in the systems provided to them, and the way those systems service their needs.

This is significant for the reason that the user will make systems decisions based on only one criterion from the developer's viewpoint. If the system adequately serves the needs of the user, even if the system is of low technical quality, the user will be hesitant to authorize expenditures for a replacement system whose quality is much higher, and which might provide better service. Similarly, if the system does not supply adequate service to the user, and will entail significant replacement costs as well as technical complexity, the user will more readily authorize the expenditure of funds in order to alleviate his immediate need for adequate service. Thus, while the developer sees the quality of the system in many lights, in terms of error rates, error tolerance, readability, ease of debugging, etc., the user perceives the system in only one way - how well it meets his needs.

To the developer, these characteristics obviously have a cumulative effect on the user's perception of the quality of the system, but for the most part user's do not have this awareness since it requires that they have some experience in the technical aspects of systems development in order to be aware of the problems associated with the task. It is the responsibility of the developer to be aware of the user's needs, his perception of what a quality system is and to develop the system in consonance with those perceptions. This often is not a simple task, since perceptions can change in time. Thus, a user may gain maturity in his appreciation of systems if exposed to "user-friendly" systems, or may become less systems oriented after being exposed to systems which are difficult to work with.

At the highest level of our quality metric framework, some of the required translation between user and developer can be accommodated. The quality factors show relevancy of technical aspects of the software to the user's needs over the life cycle of the system. Utilizing the factors, the user can appreciate the impact of a system which is unreliable, or hard to maintain, or hard to change. The user sees this impact in terms of cost, the user's ultimate measure, and its effect on the service the system provides.

Thus, it is beneficial to organizations to view software developers as providers of services over long period of time. Many large management information systems are developed with planned life spans of ten years. The enchancement or rewrite of such systems are large undertakings requiring the investigation of down-stream processing impact. If one views software systems as services, then one can view such problems in light of their impact on the provision of service to users and customers, thus on the entire organization, rather than on individual modules or subsystems. This enables managers to make more rational decisions based on an overall organizational viewpoint.

An important point to note, related to the nature of organizations, is that change is unavoidable. Military organizations and systems change as missions change and it follows that information systems which they use must change with them.

A corollary to this is the fact that the tendency of software managers to engage in "plant protection." i.e., trying to avoid changing software systems if at all possible (while positive in some instances) can have a detrimental impact on the overall goals of the organization.

Products go away after a time. Sometimes the life span is long, such as the B-52 or the Volkswagen "Beetle, " but eventually they are replaced by new products. The need for specific services lasts a very long time. This long life span is a problem which an awareness of the service perspective gives in the application of quality metrics. The use of metrics throughout the

the life cycle gives us a method for effectively specifying and monitoring the delivery of service to the user during the operational/maintenance phase of the life cycle.

2.3 CLASSIFICATION OF METRICS

The actual measurement of software quality is accomplished by applying software metrics (or measurements) to the documentation and source code produced during a software development. These measurements are part of the established model of software quality and through that model can be related to various user-oriented aspects of software quality.

The metrics can be classified according to three categories:

- anomaly-detecting
- predictive
- acceptance

Anomaly-detecting metrics identify deficiencies in documentation or source code. These deficiencies usually are corrected to improve the quality of the software product. Standards enforcement is a form of anomaly-detecting metrics.

Predictive metrics are measurements of the logic of the design and implementation. These measurements are concerned with form, structure, density, and complexity type attributes. They provide an indication of the quality that will be achieved in the end product, based on the nature of tne application, and design and implementation strategies.

Acceptance metrics are measurements that are applied to the end product to assess the final compliance with requirements. Tests are a form of acceptance-type measurements.

The measurements contained in Appendix B are either anomaly-detecting or predictive metrics. They are applied during the development phases to assist in identification of quality problems early so that corrective actions can be taken early when they are more effective and economical.

The measurement concepts complement current Quality Assurance and testing practices. They are not a replacement for any current techniques utilized in normal quality assurance programs. For example, a major objective of quality assurance is to assure conformance with user/customer requirements. The software quality metric concepts described in this manual provide a methodology for the user/customer to specify life-cycle-oriented quality requirements, which are usually not considered, and a mechanism for measuring if those requirements have been attained. A function usually performed by quality assurance personnel is a review/audit of software products produced during a software development. The software metrics add formality and quantification to these document and code reviews. The metric concepts also provide a vehicle for early involvement in the development since there are metrics which apply to the documents produced early in the development.

Testing is usually oriented toward correctness, reliability, and performance efficiency. The metrics assist in the evaluation of other qualities like maintainability, portability, and flexibility.

2.4 COMPARING METRICS, WALK-THROUGHS AND INSPECTION

Over the past ten years, a number of different quality assurance or software design methodologies have been developed in response to the problems which both the Government and the private sector have experienced in obtaining and producing quality software. In a field as complex as software engineering, the dogmatic adherence to one or another methodology is counter productive; as methodologies evolve in response to the problems, borrowing and sharing of ideas produces hybrids which will be more capable of coping with the problems.

Two methodologies related to software metrics are Code Inspections and Structured Walk-throughs. Both were originally developed to aid in what is essentially the quality assurance function. In this section we will compare them with software metrics and point out their shared strengths and weaknesses and the stages of development during which they are most effectively applied.

CODE INSPECTIONS

The Code Inspection technique was developed by Fagan [FAGM76]. It has primarily one purpose, finding errors in design or code. The methodology associated with code inspections consists of conducting a series of inspections during the software development, one at the completion of the design stage, one at the completion of coding, and subsidiary ones (e.g. publications inspections) throughout the IBM-defined levels of programming process operations.

Fagan divides the "programming process" into three different subprocesses (or "miniprocesses): Design, Code, and Test. A seperate unnamed subprocess consists of a statement of objectives. Each subprocess is divided into "Levels". There are nine levels, numbered 0 through 8. The purpose of code inspections is to control the programming process by determining when "exit criteria" for a particular subprocess or level are satisfied. The major inspections occur at the completion of Design and Code subprocesses. "Reworking" of the unit when errors are found in any level or subprocess must be done before it can be claimed to be completed.

Code inspections concentrate on error detection and correction through a formally defined "process-control" methodology.

WALK-THROUGHS

Design Walk-Throughs have no agreed upon structure common to all groups which make use of them. In some installations, the Walk-Through is structured like the Code Inspection, in others there is little or no structure. The primary idea, however, is the same everywhere, and that is peer-review of the system design and coding. Walk-Throughs can be conducted at both the design stage and during coding.

These peer-reviews ane generally conducted as a team meeting, with representatives of the designers (and coders in a code walk-through), sometimes management and users. The purpose is to subject the design (or code) to a critical evaluation.

IBM defines eight basic characteristics to the Walk-Through [IBM 74]:

- Arranged and scheduled by developer
- Not used for employee evaluation
- Participants include all involved areas
- Should have a defined set of attainable objectives
- Review materials distributed in advance of meeting and reviewers should come prepared with questions
- Roles of reviewers and tasks to be performed are known to participants
- A moderator controls the course of the Walk-Through and compiles the list of errors and inconsistencies to be acted on
- Problem resolution takes place outside of the Walk-Through

INSPECTIONS, WALK-THROUGHS AND METRICS

The application of software metrics is not meant to supplant useful methods of quality assurance such as Code Inspection and Walk-Through, but to be used in *conjunction with them* as part of an intergrated program. Software metrics as we have noted previously have both anomaly-detecting and predictive characteristics, in addition to those which may be classfied as acceptance metrics.

Code Inspections and Walk-Throughs are oriented towards anamoly-detection. They can be very useful during certain phases of the development life cycle. They are development team techniques. The metrics, on the other hand, not only can be used by the development team but also can be used by the acquisition manager as acceptance criteria. A complete comparision between metrics, walk-throughs, and code inspections is shown in Table 2.4-1. Part of this table was excerpted from [FAGM 76]. A comparision of walk-throughs and code inspections based on a classroom experiment was presented in [MYEG 78].

TABLE 2.4-1

Comparison of Key Properties of Inspections and Walk Thrus and Metrics

Properties	Inspection	Walk-Thru	Metrics
1. Formal Moderator Training	Yes	No	No
2. Definite Participant Roles	Yes	No	Yes
3. Who "Drives" The Insp. or W-T	Moderator	Owner of Material (Designer or Coder)	Quality Assurance Group
4. Use "How to Find Errors" Checklists	Yes	No	Yes
5. Use Distribution of Error Types to Look For	Yes	No	Yes
6. Follow-Up to Reduce Bad Fixes	Yes	No	Yes
7. Less Future Errors Because of Detailed Error Feedback to Individual Programmer	Yes	Incidental	Yes
8. Improve Inspection Efficiency From Analysis of Results	Yes	No	Yes
9. Analysis of Data Process Problems Improvements	Yes	No	Yes
10. Lifecycle Impact and Applicability?	Partial	No	Yes
11. Quantification of Results For Comparative Purposes	No	No	Yes
12. Prediction of Quality Level Based on Current Analysis and Figure of Merit?	No	No	Methodology Exists
13. Formal Definition of Quality (Factors, Attributes)?	No	No	Yes
14. Formal Validation of Concept Carried Out?	Partial (lack of quantifiable results makes it difficult to statistically validate)	No	Yes
15. Formal Methodology for Application Developed?	Yes	No	Yes
16. Applicable in Different Environments	Yes	Yes	Yes

2.5 METRICS AS A QUALITY ASSURANCE MIS

Current Quality Assurance programs include configuration management systems which control versions of the software and modifications to baselines. A problem report control system is usually a part of many of these systems with which problem reports are formally logged, reported, and closed. These tools have aided quality assurance personnel not only by providing automated support to activities that are performed during a development but also from a historical viewpoint. They provide a data base from which analyses of error types and error rates can be made. Based on these analyses, the emphasis of testing can be changed during future developments.

In a similar manner, the metrics data provides a profile of the technical aspects of the software. Such data as number of lines of code, number of comments, number of paths through a module, etc. also provides a data base from which analyses can be made to better orient the quality assurance program to controlling the quality of the software produced. The control is imparted by audits, standards and conventions, and tests. Emphasis in each of these activities can be oriented based on past development experiences. The metric data base base provides hard data upon which to base the reorientation rather than basing it on subjective feelings.

Thus the metric data are not only useful during a development as indicators of the quality being achieved but also complement configuration management and problem history data as a quality assurance management information system. The retention of the metric data in machine readable form is a key to the utility of the data for this purpose.

SECTION 3

ANALYSIS OF METRIC CONCEPTS IN O HER ENVIRONMENTS

3.1 APPROACH

The analysis of the applicability of the software quality metrics to environments other than the command and control environment in which the metrics were intially developed is a two-step process. The first step involves an assessment of the new environment and its real attributes (system life cycle, users needs, and development environment), a comparison or derivation of an analogy between these real attributes with those of the C^2 environment, and an evaluation of how well the model of software quality represented by our framework and definitions fit this new environment. This process is shown in figure 3.1-1.

The evaluation process represents the derivation of an hypothesized analogy between the model of software quality developed for the C^2 environment and a broader, more general model (the refined model) which subsumes both the C^2 and MIS environments. This latter model has its basis in certain observed characteristics (real attributes), which are common to both the C^2 and MIS environments. This analysis was performed as Task 1 of this research effort and is the subject of this section of the report. In actually conducting this evaluation, we proceeded by analyzing the applicability of each factor, then each criterion and finally each metric. The number of changes made to the model will be shown to be minimal, primarily because a goal of the previous effort was to establish metrics which were language independent.

The second step was performed during the final phase of the research effort. This step involved the direct application of the same methodology used during the previous effort to apply the metrics and mathematically validate their correlation with the qualities of the system as represented by its operational and maintenance historical data. The results of this step are contained in the next section.

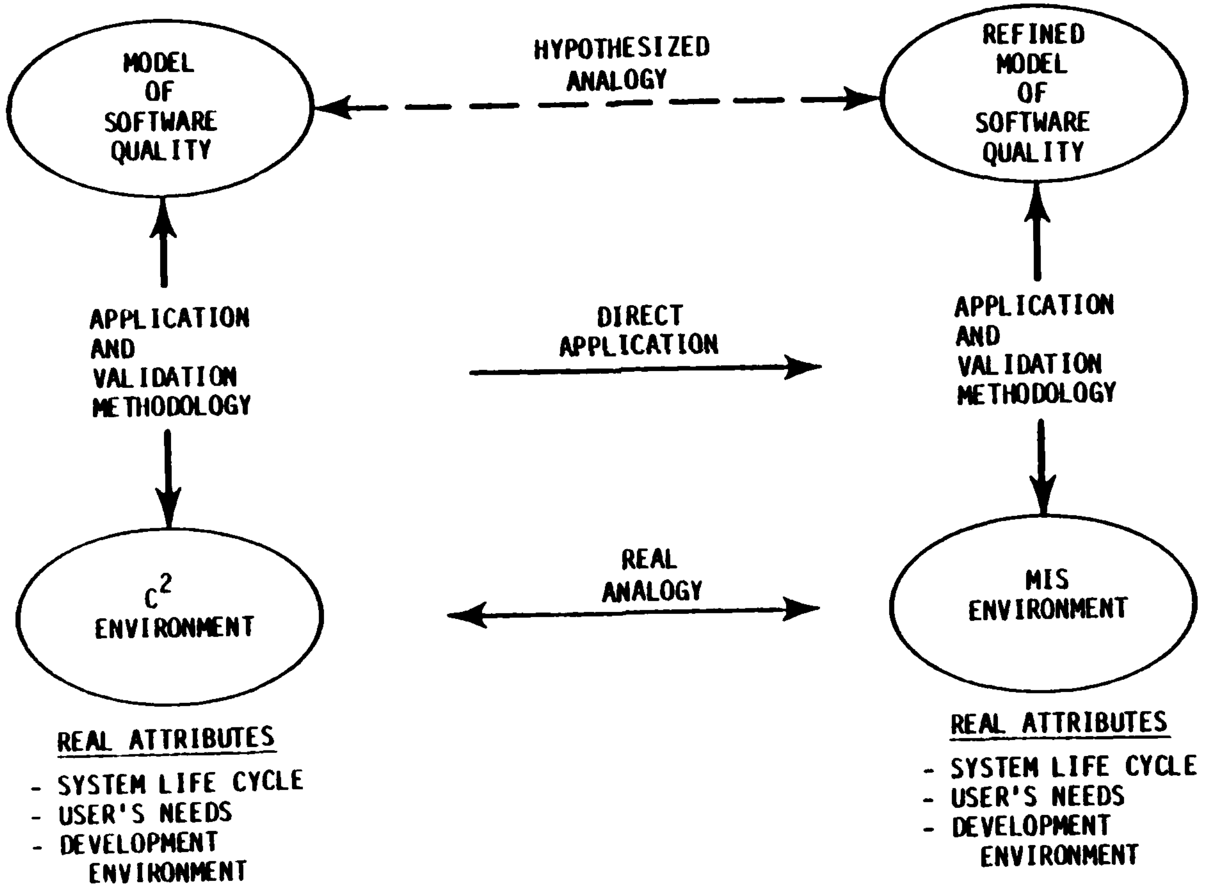


Figure 3.1-1 Evaluation Process

1760B

3.2 SOFTWARE QUALITY REQUIREMENTS SURVEY

In order to perform a thorough evaluation of the applicability of the software quality metrics to other development environments, each level of the framework must be investigated in light of the peculiarities of each particular environment. This section describes the approach and the results of our evaluation at the quality factors level.

At this level, an evaluation of the applicability of the quality factor to the particular environment is necessary. Each of the eleven quality factors were evaluated with respect to the Army Computer Systems Command development environment and particularly the development of the MARDIS system. There were no indications that any new factors were necessary. Each of the current factors seemed applicable. In evaluating the support software system, the same conclusions were made.

It would be naive, however, to expect that those factors which are critical to one environment would be equally critical to another. In fact, much can be learned about a system and the usefulness of the software quality metrics concepts by simply looking at the differences between systems in light of the quality factors. For example, in paragraph 3.2 of RADC TR-77-369, the importance of the individual quality factors was discussed in relationship to various examples of systems, ranging from testbed or R&D laboratory systems to airborne avionics and manned spacecraft. To accomplish this evaluation during this effort, a brief survey (see table 3.2-1) was provided to personnel at the Air Force Electronics Systems Division (ESD). The intent of the survey was to solicit the viewpoints of personnel concerning which quality factors are important to the particular system on which they are currently working. Most responses from ESD involved indications and warning systems.

Table 3.2-1 Software Quality Requirements Survey Form

1. The 11 quality factors listed below have been isolated from the current literature. They are not meant to be exhaustive, but to reflect what is currently thought to be important. Please indicate whether you consider each factor to be Very Important (VI), Important (I), Somewhat Important (SI), or Not Important (NI) as design goals in the system you are currently working on.

RESPONSE	FACTORS	DEFINITION
________	CORRECTNESS	Extent to which a program satisfies its specifications and fulfills the user's mission objectives.
________	RELIABILITY	Extent to which a program can be expected to perform its intended function with required precision.
________	EFFICIENCY	The amount of computing resources and code required by a program to perform a function.
________	INTEGRITY	Extent to which access to software or data by unauthorized persons can be controlled.
________	USABILITY	Effort required to learn, operate, prepare input, and interpret output of a program.
________	MAINTAINABILITY	Effort required to locate and fix an error in an operational program.
________	TESTABILITY	Effort required to test a program to insure it performs its intended function.
________	FLEXIBILITY	Effort required to modify an operational program.
________	PORTABILITY	Effort required to transfer a program from one hardware configuration and/or software system environment to another.
________	REUSABILITY	Extent to which a program can be used in other applications - related to the packaging and scope of the functions that programs perform.
________	INTEROPERABILITY	Effort required to couple one system with another.

2. What type(s) of application are you currently involved in?

____

3. Are you currently in:

________ 1. Development phase
________ 2. Operations/Maintenance phase

4. Please indicate the title which most closely describes your position:

________ 1. Program Manager
________ 2. Technical Consultant
________ 3. Systems Analyst
________ 4. Other (please specify)________

The survey sheet that was sent to ESD was also distributed to several projects in process at our location. These projects represent command and control, support software, and simulation applications.

Thirty-nine responses were received. The profile of these responses by type of application is shown in table 3.2-2

Table 3.2-2 Response Type Profile

NO.	TYPE	COUNT
1	SUPPORT SOFTWARE	6
2	SIMULATION	5
4	COMMAND & CONTROL	4
6	INDICATIONS & WNG	24

The responses were grouped by type of application initially. For each type, the responses were summed, averaged and a standard deviation for the range of responses for the ratings of the quality factors was calculated. Factors were rated very important, important, somewhat important, or not important as design goals by each respondee. These ratings were given the values four (4) through one (1) respectively.

Histograms of the responses by type of application were also generated for visual comparison between types.

The responses were then grouped by the phase the responder had participated in with respect to the subject system. For each phase the sum, average and standard deviation of the responses were calculated. Twenty responses were grouped in the development phase and 19 were grouped in the maintenance phase.

Histograms were generated for comparison between the two phases.

The results of the analysis are given in the following paragraphs.

3.2.1 SUPPORT SOFTWARE

Tables 3.2-3 and 3.2-4 give the sum, average and standard deviation for 6 responses about a software development support system which has been developed in Sunnyvale. It is operational and has been distributed to a number of different sites on a number of different hardware configurations. The concern for usability reflects the requirement that the system will be "user-friendly" enough to be an effective development tool. Similarly, the concern for portability represents the difficulty of the conversion process and the desire to transfer the system to a number of GE development environments. Reusability is important in this environment because it is part of an R&D effort in which prototype tools are developed and evaluated. Those capabilities which users identify as worthwhile are kept and further developed into a final product. The ability to "reuse" parts of the software is important.

Table 3.2-3 Summary By Type - Support Software
Sum of Scores For 6 Samples

USABILITY	23
RELIABILITY	21
MAINTAINABILITY	21
PORTABILITY	21
CORRECTNESS	20
REUSABILITY	20
FLEXIBILITY	19
TESTABILITY	15
INTEROPERABILITY	15
EFFICIENCY	10
INTEGRITY	8

Table 3.2-4 Summary By Type - Support Software
Average of Scores For 6 Samples With STD Deviation

USABILITY	3.83	0.41
RELIABILITY	3.5	0.55
MAINTAINABILITY	3.5	0.55
PORTABILITY	3.5	0.84
CORRECTNESS	3.33	0.52
REUSABILITY	3.33	0.82
FLEXIBILITY	3.17	0.41
TESTABILITY	2.5	0.55
INTEROPERABILITY	2.5	1.22
EFFICIENCY	1.67	0.82
INTEGRITY	1.33	0.82

3.2.2 SIMULATION

Tables 3.2-5 and 3.2-6 provide the results of 5 survey responses for a simulation system developed in Sunnyvale. The system is utilized as a planning tool by analysts. The high requirements for correctness and reliability represent the fact that the output of the simulation system assumes a critical role in the planning and operation of the actual system being simulated. The concern for usability reflects that system's use by analysts and the importance given to its interaction with customer-users.

Table 3.2-5 Summary By Type - Simulation
Sum of Scores For 5 Samples

CORRECTNESS	20
RELIABILITY	20
USABILITY	18
MAINTAINABILITY	16
TESTABILITY	14
FLEXIBILITY	14
EFFICIENCY	12
INTEGRITY	10
INTEROPERABILITY	9
PORTABILITY	7
REUSABILITY	5

Table 3.2-6 Summary By Type - Simulation
Average of Scores For 5 Samples With STD Deviation

CORRECTNESS	4	0
RELIABILITY	4	0
USABILITY	3.6	0.55
MAINTAINABILITY	3.2	0.45
TESTABILITY	2.8	0.45
FLEXIBILITY	2.8	1.1
EFFICIENCY	2.4	0.55
INTEGRITY	2	1.41
INTEROPERABILITY	1.8	1.3
PORTABILITY	1.4	0.55
REUSABILITY	1	0

The simulation system is an operational system. Thus maintainability is also an important concern. In addition, for certain analyses, modifications are required. Therefore flexibility and testability are also important.

3.2.3 COMMAND AND CONTROL

Tables 3.2-7 and 3.2-8 provide the results of the four responses related to satellite command and control systems which have been developed or are being developed in Sunnyvale. The fact that correctness, reliability, and testability are all rated very high tends to reflect the fact that the C^2 software is critical to the success of the mission of the system. The systems, once developed, are also maintained by GE for the Air Force, undergo major revisions and operate on relatively small (in Storage) machines. These characteristics are represented by the high rankings given flexibility, efficiency, and maintainability.

Table 3.2-7 Summary By Type - Command & Control
Sum of Scores For 4 Samples

CORRECTNESS	16
RELIABILITY	15
TESTABILITY	15
FLEXIBILITY	13
EFFICIENCY	12
MAINTAINABILITY	12
USABILITY	10
INTEGRITY	8
REUSABILITY	6
INTEROPERABILITY	6
PORTABILITY	5

Table 3.2-8 Summary By Type - Command & Control
Average of Scores For 4 Samples With STD Deviation

CORRECTNESS	4	0
RELIABILITY	3.75	0.5
TESTABILITY	3.75	0.5
FLEXIBILITY	3.25	0.5
EFFICIENCY	3	0.82
MAINTAINABILITY	3	0.82
USABILITY	2.5	0.58
INTEGRITY	2	1.15
REUSABILITY	1.5	1
INTEROPERABILITY	1.5	1
PORTABILITY	1.25	0.5

3.2.4 INDICATIONS AND WARNING

Tables 3.2-9 and 3.2-10 present the results of the 24 survey responses received from ESD. The results are similar to the Command and Control software except for the high ratings given interoperability and usability. The requirements for effective man-machine interaction and system to system interaction so important to an indications and warning system is expressed by these ratings.

Table 3.2-9 Summary by Type - Indications and Warning
Sum of Scores for 24 Samples

CORRECTNESS	93
RELIABILITY	90
MAINTAINABILITY	75
INTEROPERABILITY	73
USABILITY	72
TESTABILITY	72
FLEXIBILITY	70
INTEGRITY	67
EFFICIENCY	66
PORTABILITY	46
REUSABILITY	41

Table 3.2-10 Summary by Type - Indications and Warning
Average of Scores for 24 Samples with STD Deviation

CORRECTNESS	3.88	0.45
RELIABILITY	3.75	0.53
MAINTAINABILITY	3.13	0.85
INTEROPERABILITY	3.04	1.08
USABILITY	3	0.88
TESTABILITY	3	0.88
FLEXIBILITY	2.92	0.88
INTEGRITY	2.79	1.02
EFFICIENCY	2.75	0.85
PORTABILITY	1.92	0.97
REUSABILITY	1.71	0.75

To facilitate comparison of the expressed quality requirements by type, histograms for each type are provided in Figures 3.2-1, 2, 3, and 4.

Figure 3.2-1 Histogram of Average Scores by Type - Support Software
6 Samples

```
4-                          ._.
             ._.            |/|  ._.            ._.
        ._.  |/|            |/|  |/|       ._.  |/|  ._.
        |/|  |/|            |/|  |/|       |/|  |/|  |/|
3-      |/|  |/|            |/|  |/|       |/|  |/|  |/|
        |/|  |/|            |/|  |/|  ._.  |/|  |/|  |/|  ._.
        |/|  |/|            |/|  |/|  |/|  |/|  |/|  |/|  |/|
        |/|  |/|            |/|  |/|  |/|  |/|  |/|  |/|  |/|
2-      |/|  |/|  ._.       |/|  |/|  |/|  |/|  |/|  |/|  |/|
        |/|  |/|  |/|       |/|  |/|  |/|  |/|  |/|  |/|  |/|
        |/|  |/|  |/|  ._.  |/|  |/|  |/|  |/|  |/|  |/|  |/|
        |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|
1-      |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|
        |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|
        |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|
        |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|
-----------------------------------------------------------------
        COR  REL  EFF  INT  USA  MNT  TES  FLX  PRT  REU  IOP
```

Figure 3.2-2 Histogram of Average Scores by Type - Simulation
5 Samples

```
        ._.  ._.
4-      |/|  |/|
        |/|  |/|            ._.
        |/|  |/|            |/|  ._.
        |/|  |/|            |/|  |/|
3-      |/|  |/|            |/|  |/|  ._.  ._.
        |/|  |/|  ._.       |/|  |/|  |/|  |/|
        |/|  |/|  |/|       |/|  |/|  |/|  |/|
        |/|  |/|  |/|  ._.  |/|  |/|  |/|  |/|
2-      |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|            ._.
        |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  ._.       |/|
        |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|       |/|
        |,:  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  ._.  |/|
1-      |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|
        |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|
        |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|
        |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|  |/|
-----------------------------------------------------------------
        COR  REL  EFF  INT  USA  MNT  TES  FLX  PRT  REU  IOP
```

Figure 3.2-3 Histogram of Average Scores by Type - Command & Control
4 Samples

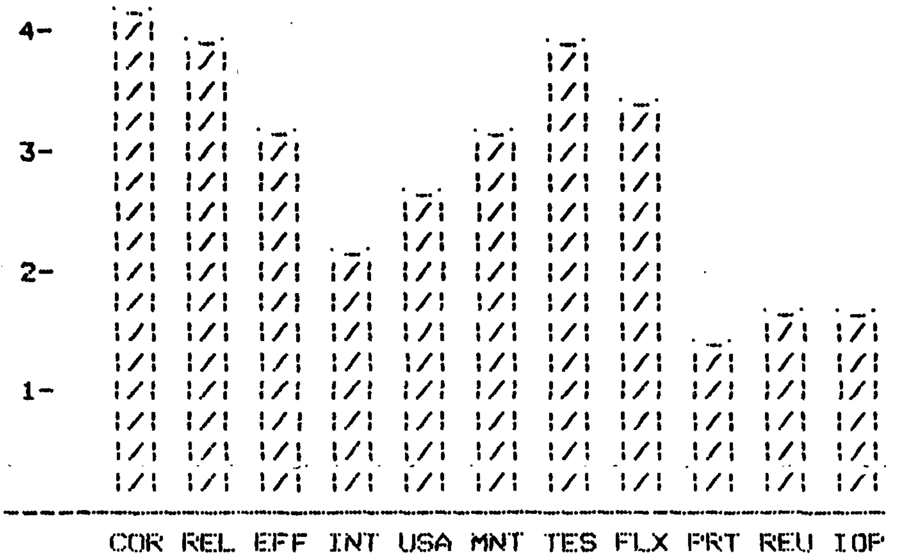
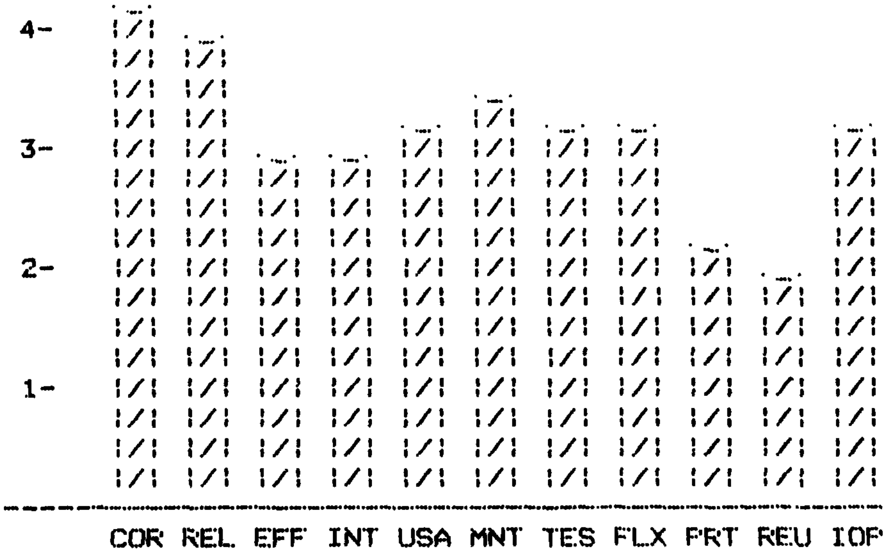


Figure 3.2-4 Histogram of Average Scores by Type - Indications & WNG
24 Samples



3.2.5 DEVELOPMENT PHASE

Tables 3.2-11 and 3.2-12 provide the statistics of 20 responses spanning all of the previously mentioned application types. These responses were from personnel involved in the development of the system. Note the concern for testability which to some extent reflects their concern for the immediately succeeding phases of the development.

Table 3.2-11 Summary By Phase - Development
Sum of Scores For 20 Samples

CORRECTNESS	80
RELIABILITY	77
TESTABILITY	62
INTEROPERABILITY	60
MAINTAINABILITY	59
USABILITY	58
FLEXIBILITY	58
EFFICIENCY	54
INTEGRITY	54
PORTABILITY	36
REUSABILITY	34

Figure 3.2-12 Summary By Phase - Development
Average Scores For 20 Samples With STD Deviation

CORRECTNESS	4	0
RELIABILITY	3.85	0.49
TESTABILITY	3.1	0.91
INTEROPERABILITY	3	1.08
MAINTAINABILITY	2.95	0.89
USABILITY	2.9	0.91
FLEXIBILITY	2.9	0.72
EFFICIENCY	2.7	0.73
INTEGRITY	2.7	1.03
PORTABILITY	1.8	0.89
REUSABILITY	1.7	0.73

3.2.6 MAINTENANCE PHASE

Tables 13 and 14 provide the statistics of 19 responses, again from a number of different types of applications, for personnel involved in the operations and maintenance phase. Note the concern in this phase focuses on the maintainability, usability, and flexibility of the software.

Table 3.2-13 Summary by Phase - Maintenance and Operations
Sum of Scores for 19 Samples

CORRECTNESS	69
RELIABILITY	69
USABILITY	65
MAINTAINABILITY	65
FLEXIBILITY	58
TESTABILITY	54
EFFICIENCY	46
PORTABILITY	43
INTEROPERABILITY	43
INTEGRITY	39
REUSABILITY	38

Table 3.2-14 Summary by Phase - Maintenance and Operation
Average Scores For 19 Samples With STD Deviation

CORRECTNESS	3.63	0.6
RELIABILITY	3.63	0.5
USABILITY	3.42	0.69
MAINTAINABILITY	3.42	0.51
FLEXIBILITY	3.05	0.91
TESTABILITY	2.84	0.69
EFFICIENCY	2.42	1.02
PORTABILITY	2.26	1.24
INTEROPERABILITY	2.26	1.28
INTEGRITY	2.05	1.22
REUSABILITY	2	1.2

For comparison between phases, Figures 3.2-5 and 3.2-6 are provided.

Figure 3.2-5 Histogram of Average Scores by Phase - Development
20 Samples

```
    ._.
4-  |/| ._.
    |/| |/|
    |/| |/|
    |/| |/|                 ._. ._. ._. ._.             ._.
3-  |/| |/| ._. ._. |/| |/| |/| |/|                     |/|
    |/| |/| |/| |/| |/| |/| |/| |/|                     |/|
    |/| |/| |/| |/| |/| |/| |/| |/|                     |/|
    |/| |/| |/| |/| |/| |/| |/| |/|                     |/|
2-  |/| |/| |/| |/| |/| |/| |/| |/| ._. ._. |/|
    |/| |/| |/| |/| |/| |/| |/| |/| |/| |/| |/|
    |/| |/| |/| |/| |/| |/| |/| |/| |/| |/| |/|
    |/| |/| |/| |/| |/| |/| |/| |/| |/| |/| |/|
1-  |/| |/| |/| |/| |/| |/| |/| |/| |/| |/| |/|
    |/| |/| |/| |/| |/| |/| |/| |/| |/| |/| |/|
    |/| |/| |/| |/| |/| |/| |/| |/| |/| |/| |/|
    |/| |/| |/| |/| |/| |/| |/| |/| |/| |/| |/|
-----------------------------------------------
    COR REL EFF INT USA MNT TES FLX PRT REU IOF
```

Figure 3.2-6 Histogram of Average Scores by Phase - Maintenance & Operations
19 Samples

```
4-  ._. ._.
    |/| |/|         ._. ._.
    |/| |/|         |/| |/|
    |/| |/|         |/| |/|     ._.
3-  |/| |/|         |/| |/| ._. |/|
    |/| |/| ._.     |/| |/| |/| |/|
    |/| |/| |/|     |/| |/| |/| |/| ._.     ._.
    |/| |/| |/| ._. |/| |/| |/| |/| |/| ._. |/|
2-  |/| |/| |/| |/| |/| |/| |/| |/| |/| |/| |/|
    |/| |/| |/| |/| |/| |/| |/| |/| |/| |/| |/|
    |/| |/| |/| |/| |/| |/| |/| |/| |/| |/| |/|
    |/| |/| |/| |/| |/| |/| |/| |/| |/| |/| |/|
1-  |/| |/| |/| |/| |/| |/| |/| |/| |/| |/| |/|
    |/| |/| |/| |/| |/| |/| |/| |/| |/| |/| |/|
    |/| |/| |/| |/| |/| |/| |/| |/| |/| |/| |/|
    |/| |/| |/| |/| |/| |/| |/| |/| |/| |/| |/|
-----------------------------------------------
    COR REL EFF INT USA MNT TES FLX PRT REU IOF
```

These results substantiate the hypothesis that quality concerns relate to the life cycle phase through which the system is passing and the application environment in which the system resides. These conclusions support the framework that has been developed on two counts:

1. Since different applications have different quality needs, a framework is needed in which those needs or goals can be identified, the interrelationships between quality goals recognized, and the progress toward achieving those goals monitored.
2. Since the perspective of quality needs changes over the life cycle of a system, a necessary attribute of the quality framework is a life cycle view of quality. This attribute forces consideration of long range quality goals as well as shorter range goals at the beginning of the systems life.

3.3 APPLICABILITY OF CRITERIA IN OTHER ENVIRONMENTS

The establishment of criteria for software quality factors had a fourfold purpose - to further define each factor, to describe relationships between each factor, to establish a unique correspondence between metrics and criteria, and finally to preserve the hierarchical framework of the factors. In determining whether the criteria previously established are applicable in new environments particularly in the MIS environment, it is necessary to shuw that the refined definitions of factors using criteria, the relationships between factors, and the correspondence between metrics and criteria are preserved by our hypothesized analogy between models of software quality.

Criteria are software attributes or characteristics which contribute to software quality factors. Whereas quality factors are management-oriented views of software, criteria are software-oriented. To show that the same criteria hold in non-C^2 environments, it must be shown that the same software attributes are present in other environments and that they still contribute to the same quality factors.

In order to show that each software attribute was present in other environments, each criterion was analyzed to determine if it could be shown to exist in software in other environments. But since these criteria are very general, system-level attributes of software, it was seen immediately that non-C^2 software must also share these characteristics. Like factors, criteria are "about" software, and grouping them in order to preserve a specific quality factors definition is natural.

To determine if the same relationships between the defining criteria and the factors held, an analysis of figure 1.3-3 was conducted. Again the results indicated that these relationships were preserved in other environments.

The analysis of the correspondence between metrics and criteria is described in Section 3.4. The results of this analysis indicated that except for a few very minor points, the same set of metrics held in the non-C^2, MIS environment.

3.4 REVIEW OF THE METRICS

The result of our review of the quality metrics are in Appendix B. Some examples of the analysis performed and the rationale for modifying, deleting, or adding metrics are given in the paragraphs that follow. The examples are grouped logically. Paragraph 3.4.2 has examples of metrics which apply directly to the COBOL/MIS environment. Paragraph 3.4.2 has examples of metrics which, because they are not applicable to the new environment, were deleted. They may also have been deleted because they were found to be too hard to practically measure. They were only deleted if modification was not possible. Paragraph 3.4.4 has examples of metrics which were modified. The modifications in many cases represented wording changes to make what was being measured clearer or in some cases, to make the metric a relative quantity metric rather than a checklist type metric. These changes were based primarily on the additional experience gained using and applying the metrics during this study. The other reason modifications were made were to make the metric applicable to COBOL if it was not previously.

Paragraph 3.4.5 has examples of additional metrics which were considered during this effort. The validation procedure will reveal their correlation or value and consideration of retaining them was based on that.

3.4.1 QUALITY METRICS IN DIFFERENT ENVIRONMENTS

Measurements of quality, unlike fundamental measurements such as length, mass and heat; i.e., extensive properties of things, are ordinal and relative in nature. There are, however, other types of measurements, less rigorous, bxt nonetheless useful. A sphygmomanometer does not measure a fundamental, extensive property of human beings, but blood pressure is an important indicator of general health. Measurements such as these are called pointer measurements. In a certain sense, the "mechanism or procedure is made to supercede the intuition, and is used to define the property which it purports to measure" [STAR73].

Software Quality Metrics are pointer measurements. They do not purport to measure fundamental properties of software, but instead, characteristics which provide indication of the quality of the software. The past history of engineering reflects similar considerations. The desire for uniform quality is based on practical economic reasoning linked with the need for rationalized processes in mass production. Thus, uniform quality is based on a tower whose top level is its goal of quality, the lower levels being standardization of components, measurement of those components, definition of proper units of measurement, and whose foundations are fundamental concepts [NOBD77].

The metrics developed during previous research consist of phased sets of measurements and evaluations undertaken during the development phase of the life cycle. These are applicable during the requirements analysis, design and implementation phases of the life cycle. During the requirements analysis phase, 25 elements of measurement are gathered; during the design phase, 108 elements are gathered; during the implementation phase, 157 are gathered [McCJ77]. The elements of measurement are specific characteristics that are measured. A number of elements may comprise a metric. While the number of elements to be gathered may seem excessive, they do serve to give a very complete profile of the project. Part of the continuing research in the area of software quality engineering has been to develop prototype automated tools for the gathering of these measurements, and automated methods for their analysis [LOPC78]. The automated tools alleviate the high cost of manual application of the metrics and assist in the accurate, consistent application of the metrics.

Because of the fact that all of the different applications (MIS, C^2, and Support Software) are developed in basically the same phased approach, i.e., requirements analysis, design, implementation, and test, the phase or progressive application concept of the metrics is relevant to all of them.

The technique or automated tool used to take the measurements, however, will differ depending on what the documentation requirements and formats are and what programming language is used. The measurement applied during the requirements analysis and design phases are independent of the language since they are oriented toward the documentation rather than the language. Some metrics relating to design measures could be influenced by the particular implementation language. This possibility was checked for in our evaluation. The metrics applied during implementation were the most likely to be language dependent. Emphasis during our evaluation was placed in this area. The major concentration then was in evaluating the applicability of the metrics to MIS design and COBOL programming practices.

It is important to note a difference between a metric being inapplicable to COBOL, in which case it is not a generally applicable metric and is either deleted or modified, and a metric not applicable for a particular situation. An example of the latter case is the metric, RECOVERY FROM DEVICE ERRORS (ET.5). If the operating system provides facilities with this capability, perhaps with a checkpoint/restart capability, and the system does not have a critical timeline in which to function, then this metric is unimportant or not applicable to this situation. This does not mean it has no meaning in an MIS environment. It can be applied and does have meaning in certain circumstances.

3.4.2 EXAMPLES OF METRICS APPLICABLE TO COBOL

Over 90% of the metrics established in the previous research effort were determined to be applicable to the COBOL/MIS environment. Table B-1, Appendix B, is a table of metrics organized by life cycle phase and relationship to criteria and subcriteria. The organization of this table is based on the recognition that the main source of error in system development is translation from one phase to another. Thus, the three main development phases are indicated, as well as the criterion/subcriterion applicable to a factor. The elements of measurement occur in the center under the heading "METRIC."

One of the criteria of the factor reliability is accuracy. There are five measurement elements associated with this criterion. These are reproduced in Table 3.4.2-1 which is excepted from Appendix B.

Table 3.4.2-1 Accuracy Metric

FACTOR(S): RELIABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
ACCURACY	AC. 1 ACCURACY CHECKLIST:						
	(1) Error analysis performed and budgeted to module.	□					
	(2) A definitive statement of requirement for accuracy of inputs, outputs, processing, and constants.	□					
	(3) Sufficiency of math library.			□			
	(4) Sufficiency of numerical methods.			□		□	
	(5) Execution outputs within tolerances.					□	
	SYSTEM METRIC VALUE: Score total from applicable elements / # applicable elements		□		□		□

Table 3.4.2-2 illustrates the analysis performed to assess how applicable this metric is in the MIS/COBOL environment.

Initially, it was thought the metrics oriented toward accuracy would not be as applicable to MIS systems as C^2 systems. However, when consideration is given to the possible impacts of inaccuracies in a financial accounting system or a critical item inventory system, the importance of those measurements are quickly recognized.

AC.1 (1) and (2) simply are checking in a requirements document for recognition of accuracy requirements. Without a stated requirement, the likelihood of the provision of the required accuracy is less.

AC.1 (3) is insuring that consideration for accuracy requirements are not only applied to software being developed but also to "off-the-shelf" software or mathematical routines provided in the form of a library of routines by a vendor. Another situation, probably more typical of MIS development organizations, are libraries of routines developed in-house for typical calculatory functions. For example, a large insurance firm may use a library

Table 3.4.2-2 Accuracy Metric Analysis

ELEMENT	PHASE	INTERPRETATION
AC.1 (1) Error analysis performed and budgeted to module.	REQMTS	By "error analysis", we mean the amount of error which the user is willing to tolerate in the performance of a particular function. Iy may or may not be possible to budget this to a specific module at this early stage. The main objective is to analyze the error which is tolerable in the function.
AC.1 (2) A definitive statement of requirement for accuracy of inputs, outputs, processing, and constants.	REQMTS	Subsidiary to the error analysis, and concurrent to it, the developer and user should agree on the amount of accuracy they wish in the inputs, outputs, processing (which error analysis should uncover), and constants.
AC.1 (3) Sufficiency of math library	DESIGN	Often routines provided in a math library are used. These library functions should be checked during design for compliance with accuracy requirements.
AC.1 (4) Sufficiency of numerical methods.	DESIGN IMPL	Having performed the error analysis in AC.1 (1) and AC.1 (2) it is necessary during design and implementation to satisfy those needs.
AC.1 (5) Execution outpus within	IMPL	During debugging and testing in implementation it should be noted whether the analysis-set tolerances are satisfied by the outputs.

of "earning" routines to calculate the complicated question of how much of a policy has been "earned" over periods of time. For each new application, the accuracy of these routines should be checked for compliance with overall requirements. Calculation of inventory reorder points or budgetary balance also fall into this category.

AC.1 (4) is the check that the design and implementation of the algorithm satisfies the requirements. Situations to emphasize in this area might be round-off errors, number of significant digits, reorder point calculations, etc.

AC.1 (5) is a check that during debug and unit test, outputs are checked for compliance with accuracy tolerances. Here report formats as well as calculations can influence the accuracy of the outputs.

Thus each element of measurement within the accuracy metric has significance and is applicable to an MIS environment.

As a more specific example of how the metrics apply to COBOL, one of the measurement elements (element (5) in table 3.4.2-3) of the Code Simplicity Metric (SI.4) is a check that the module is not self-modifying. In a COBOL program, this check would be for the use of an ALTER Statement.

Table 3.4.2-3 Code Simplicity Metric

CODE SIMPLICITY	SI. 4 MEASURE OF CODING SIMPLICITY (by module)						
	(1) Module flow top to bottom.					□	
	(2) Negative Boolean or complicated compound Boolean expressions used. $(1 - \frac{\# \text{ of above}}{\# \text{ executable statements}})$						□
	(3) Jumps in and out of loops $\frac{(\# \text{ single entry/single exit loops})}{\text{total } \# \text{ loops}}$						□
	(4) Loop index modified $(1 - \frac{\# \text{ loop indices modified}}{\text{total } \# \text{ loops}})$						□
	(5) Module is not self-modifying.					□	
	(6) All arguments passed to a module are parametric.					□	
	(7) Number of statement labels. $(1 - \frac{\# \text{ labels}}{\# \text{ executable statements}})$						□
	(8) Unique names for variables.					□	
	(9) Single use of variables.					□	

The following example illustrates the ALTER construct in COBOL, and its use in modifying the processing in a module. A similar example is given in [YOUE72].

A COBOL Program with ALTER Statements

```
     •
     •
     •
READ-INPUT.  READ INPUT-CARDS.
     IF TRAN-CODE=0 THEN ALTER ERROR-SWITCH TO
          PROCEED TO TOTALS-PARA.
     ELSE IF TRAN-CODE=1 THEN ALTER ERROR-SWITCH TO
          PROCEED TO FINAL-PARA.
     ELSE ALTER ERROR-SWITCH TO PROCEED TO ERROR-PARA.
ERROR-SWITCH.  GO TO ERROR-PARA.
ERROR-PARA.
     MOVE TRAN-CODE TO ERROR-TYPE.
     MOVE QUANTITY TO ERROR-AMOUNT.
     MOVE DOLLAR TO ERROR-DOLLARS.
     WRITE ERROR-FILE.
     GO TO READ-INPUT.
TOTALS-PARA.
     COMPUTE TOTAL-QUANT=TOTAL-QUANT+QUANTITY.
     COMPUTE TOTAL-DOLLARS+TOTAL-DOLLARS+DOLLARS.
     MOVE TRAN-CODE TO ORDER-TYPE.
     MOVE DOLLAR TO ORDER-DOLLARS.
     MOVE QUANTITY TO ORDER-AMOUNT.
     WRITE ORDER-FILE.
     GO TO READ-INPUT.
     •
     •
     •
```

This practice introduces difficulty from both a static sense and a dynamic sense. The module is difficult to understand (static) and debugging is very difficult since the state (dynamic) of the module when an error occurs is uncertain

Another example is the Effectiveness of Comments Measure (SD.2) shown partially in table 3.4.2-4

Table 3.4.2-4 Effectiveness of Comments Measure

EFFECTIVENESS OF COMMENTS	SD. 2 EFFECTIVENESS OF COMMENTS MEASURE						
	(1) Modules have standard formated prologue comments which describe: - Module name/version number - Author - Date - Purpose - Inputs - Outputs - Function - Assumptions - Limitations and restrictions - Accuracy requirements - Error recovery procedures - References $1-\frac{\text{\# modules violate rule}}{\text{total \# modules}}$						□
	(2) Comments set off from code in uniform manner $1-\frac{\text{\# modules violate rule}}{\text{total \# modules}}$						□
	(3) All transfers of control & destinations commented $1-\frac{\text{\# modules violate rule}}{\text{total \# modules}}$						□
	(4) All machine dependent code commented $1-\frac{\text{\# modules violate rule}}{\text{total \# modules}}$						□

All of these measurements are important in COBOL. The first element, prologue comments, can be accomodated by the Identification Division and a REMARKS section. The Remarks Section should be set off as a comment, though, since it is no longer in the ANSI Standard.

Example:

```
    IDENTIFICATION DIVISION.
    PROGRAM - ID.        SAMPLE 1.
    AUTHOR.              JIM PROGRAMMER.
    INSTALLATION.        COMPUTER CENTER.
    DATE - WRITTEN.      SEPTEMBER 17, 1978.
    DATE - COMPILED.     SEPTEMBER 17, 1978.
    SECURITY.            UNCLASSIFIED.
*   REMARKS.
*            MODULE NAME/VERSION - SAMPLE 1/VERSION 2 INCLUDES MODIFICA-
*                TIONS OF 19 MAY 77, 22 JUN 77, 15 AUG 77, 7 DEC 77,
*                1 MAY 78, 17 SEP 78.
*            PURPOSE - TO PROVIDE A SAMPLE IDENTIFICATION DIVISION FOR
*                A COBOL PROGRAM WHICH COMPLIES WITH METRIC SD.2(1).
*                NOTE THAT THE IDENTIFICATION DIVISION, AND COBOL
*                DIVISION/SECTION STRUCTURE PROVIDE MUCH OF THE
*                REQUIRED INFORMATION.
*                THE ALIGNMENT SHOULD BE SUCH AS TO PROVIDE EASY
*                SCANNING.
*            INPUTS/OUTPUTS - PROVIDED IN THE INPUTS-OUTPUTS SECTION
*                BELOW.
*            ASSUMPTIONS - NONE.
*            LIMITATIONS - NONE.
*            ACCURACY REQUIREMENTS - NONE.
*            ERROR RECOVERY PROCEDURES - SHOULD BE DOCUMENTED AND
*                REFERENCED HERE.
*            REFERENCES - DOCUMENTATION TITLES SHOULD BE REFERENCED
*                HERE.
```

3.4.3 DELETED METRICS

Deletion of individual metrics was based on 3 criteria:

1. Difficulty in gathering actual measurements.
2. Inapplicability to either the COBOL language or to the MIS environment, so the metric is not generally applicable.
3. Redundancy.

We show some examples of deleted metrics below. A complete list of metrics may be found in Appendix B.

Example:

SI.1 Design Structure Measure:
(2) No duplicate functions

SI.1(2) has been deleted because in actual practice an analysis of the purpose of a function is too time-consuming and arduous a task to justify. In any case SI.1(1), SI.1(3), and SI.1(4) should adequately compensate for its elimination.

Example:

SI.1 Design Structure Measure:
(7) No global data

This measure has been deleted for two reasons. The first relates to the fact that COBOL is not a block structured language, and thus variables local to a block cannot be implemented. The second is the fact that most MIS applications are data-driven and the passing of common data from one process to another is not only necessary, but positive.

Example:

SE.1 Storage Efficiency Measure:
(3) Common data defined only once

This measure has been deleted since it is redundant with SE.1(11) below:

(11) Free of redundant data elements

$$1- \frac{\text{\# redundant data elements}}{\text{\# data elements}}$$

We believe that SE.1(11) is the more effective measure of data conciseness.

3.4.4 MODIFIED METRICS

Certain metrics have been modified to make them more applicable to the COBOL environment.

Example:

Descriptiveness of Implementation Language:

(5) One statement per line

$$1- \frac{\text{\# continuation + multiple statement lines}}{\text{total \# lines}}$$

Since COBOL is a free-form language and sentences may be sizable and still perform a useful structuring function it was decided that a statement should be interpreted in COBOL as a verb clause. Thus no penalty is assessed for a situation like the following:

Example:

```
IF IP-TRAN-CODE NOT EQUAL '0'
   OR IP-TRAN-DATE EQUALS '0178'
   MOVE ZEROS TO OP-TRAN-TYPE
   MOVE IP-TRAN-DATE TO OP-DATE.
```

Thus when applying this measure to COBOL it will be interpreted as follows:

(5) One verb clause per line

$$1-\frac{\text{\# lines having more than one verb clause}}{\text{total \# lines}}$$

Example:

MO.2 Modular Implementation Measure:

(2) All modules do not exceed standard module size (100)

$$1-\frac{\text{\# modules} > 100}{\text{total modules}}$$

The intent of this measure was to identify the number of modules which exceed the structured programming guideline for module length.

The relative difficulty which a programmer has in implementing subprograms in COBOL makes it hard to adhere to this rule since data must be passed in the same precision and length as in the calling routines. Instead, following YOURDON [YOUE76], we define a micromodule to be a named paragraph. In COBOL this measurement can be interpreted as paragraphs should not exceed 50 lines or one page of output.

The measurement has been changed to

Module Size Profile

The module length should be recorded and the length of micromodules should also be recorded in the case of a COBOL program.

Another area where our metrics deal with an important issue or characteristic but further emphasis, clarity, or sophistication is felt to be necessary is that of data bases. Two measurements, EE.3(1), data grouped for efficient processing, and EE.3(5), data indexed or referenced for efficient processing, represent a static and dynamic view of the data base area. The application of these metrics may be very difficult depending on the complexity of the system and the size of the data base itself. The additional measurements added in this area include (in EE.3):

- Size of data base
- Segmentation or compartimentalization of data base
- % of static elements (referenced but not modified) in data base
- % of dynamic elements (modified) in data base

A recent publication by McClure [McCC78] provides concepts of well-structured COBOL programs. To illustrate the consistency of our metrics with her concepts, Table 3.4.5-1 relates existing measures with six properties identified in [McCC78] as required by a well-structured program:

Table 3.4.5-1 Structured Concepts Related to Metrics

PROPERTY		RELATED METRIC
Property 1:	The program is partitioned into a set of hierarchically ordered modules.	MO.2(1)
Property 2:	The program controls structure follows a simple, hierarchical scheme.	SI.1(1) + MO.2(3) - (7)
Property 3:	Module construction is stnadardized.	SD.3(2)
Property 4:	The use of program variables in the program is made explicit.	SD.2(6)
Property 5:	Error processing follows normal control flow.	ET.1(2) - (3)
Property 6:	Well structured documentation is required in the program code.	SD.2

3.5 ARMY MIS AND AIR FORCE SOFTWARE DOCUMENTATION REQUIREMENTS SOURCES

This section defines the sources of software documentation requirements for Army Management Information Systems (MIS) and Air Force software applications.

Army MIS documentation requirements are contained in USACSCM 18-1 While the Air Force requirements come from MIL-STD-490 and DoD 4120-17M.

In defining the sources of documentation requirements we developed the following outline of generic software project phases:

- System Requirements Definition
- Functional Design
- Detailed Design
- Implementation
- Formal Testing and Verification
- Software Maintenance
- Operations

3.5.1 SYSTEM REQUIREMENTS DEFINITION

The system requirements document is the top level document in a system. The design of the entire system is based on requirements identified in this specification. The Army's MIS system requirements are documented in Volume I, Executive Summary, as required by USACSCM 18-1. For Air Force applications, the system requirements are contained in the Type A - System Specification which is required by MIL-STD-490.

3.5.2 FUNCTIONAL DESIGN

Functional design is the process of defining what software functions a system will perform, but does not address how they will be performed. For an Army MIS, functional design is documented in the General Functional Design Requirements (GFSR) and the Detailed Functional Design Requirements

(DFSR) which are part of Volume IV, Reference Material as defined in USACSCM 18-1. The Air Force follows the requirements in MIL-STD-490 for Type B5-Computer Program Development Specification. The B5 document is sometimes referred to as the Part I Specification.

3.5.3 DETAILED DESIGN

The detailed design is developed from the functional design and describes how each software functional will be performed. The detailed design for an Army MIS is contained in two volumes as described in USACSCM 18-1; Volume V, General System Analysis Documentation and Volume IV, System Program Documentation.

Detailed design documentation for the Air Force is contained in the Type C5 - Computer Program Product Specification as required by MIL-STD-490. The Type C5 document is sometimes referred to as the Part II specification.

3.5.4 FORMAL TESTING AND VERIFICATION

The Army requires some test planning documentation in the GFSR which is part of Volume IV, Reference Material, however, some classes of programs do not require formal test documentation. Chapter 5 of USACSCM 18-1 contains requirements for the documentation of test planning, test conducting, and reporting.

Air Force guidelines for documentation of the Test and Implementation Plan and the Test Analysis Report are contained in DoD 4120.17M.

3.5.5 SOFTWARE MAINTENANCE

Software maintenance includes software error correction and modifying, adding, or removing software functions.

USACSCM 18-1, Chapter 9, ADP System Maintenance defines documentation for software maintenance for an Army MIS. DoD 4120.17M describes the requirements for a Program Maintenance Manual.

3.5.6 OPERATIONS

Operations includes those functions required by an operation or system user to exercise the software system. USACSCM 18-1, Volume III, Operations and Maintenance, contains the documentation defining system operations for an Army MIS. Operations for the Air Force are documented in the Computer Operations Manual as defined by DoD 4120.17M. Volume II, User Documentation as defined in USACSCM 18-1 contains user information for an Army MIS. DoD 4120.17M defines the User Manual for Air Force Systems.

3.5.7 COMPARISON

A comparison between documentation requirements was planned in an initial report plan. At first, the comparison was to be between Army requirements and Air Force requirements. To be of more general use, it was then felt that a more meaningful comparison would be between MIS documentation requirements and C^2 documentation requirements. Neither comparison has been documented for the following reasons:

- A single comparison using only the documentation specifications (the Military Standards and CSCM 18-1) would be ambiguous since it is the interpretation and enforcement of those specifications that are important.
- A single comparison using our C^2 environment and the MARDIS environment could not be generalized and would be misleading as representative of a comparison of all AF/Army systems or of all C^2/MIS environments.
- The quality (including completeness and consistency) of the documentation is what is important. Different applications require different levels of detail in their documentation. However, certain key information should exist. The metrics which are applied to documentation are oriented toward assessing the quality and the existence of the key information.

To provide some general observations about current documentation requirements relative to the general application categories of C^2, MIS, and support software, the following points are made:

- C^2 environments have the most formal, voluminous, detailed documentation requirements. These systems are almost always developed in a government acquisition manager-development contractor environment where formal contractual requirements are levied on the developer for deliverable documentation. The requirements are usually a rigorous application of military standards with additional local requirements also imposed. In some cases, detailed outlines of what information should be provided in each document are given. The documents are milestone driven and often delivered more than once in draft, final, and updated forms.
- MIS environments have less volume and detail in documentation than C^2. The systems are more typically in-house development efforts. The USACSC, for example, is the central developer of multicommand MIS for the Army. The documentation requirements are more like formal in-house standards. The user/customer is less likely to require additional documentation other than the normal standards. The reviews and timing requirements of the documents are less strict.
- Support software, unless developed for commercial marketing, usually has only very informal documentation. At best the documentation requirements may follow some informal in-house development standards. Many support software tools are developed without a wide user population in mind or with little user interaction early in the development process. Often, a software tool will be developed as a prototype and then evaluated by users after completion of the initial development.

Appendix A provides additional information on the USACSC documents typically produced during an MIS software development. Appendix B, RADC TR-77-369, provides a description of the documents typically produced for an Air Force C^2 software development.

SECTION 4

APPLICATION AND VALIDATION OF METRICS

4.1 APPLICATION APPROACH

The metrics established in RADC TR-77-369 were applied under this contract to two systems described in the subsequent paragraphs. The operational and maintenance histories of the two systems were then used to determine the validity of the metrics as indicators of software quality. The application and validation are described in this section.

4.1.1 MARDIS OVERVIEW

The Modernized Army Research and Development Information System (MARDIS) is a vertical management information system. MARDIS supnorts the program formulation, phase schedule, and budget apportionment processes in R&D through the processing of resource, performance, and milestone data.

MARDIS assists the R&D community by providing timely, accurate,and consistent management information dealing with the Army's RDTE program. The source of most of the data in MARDIS originates from the laboratory scientist, technician, or engineer engaged in research and development. The information includes performance, schedule, and resource data. The system takes information once included in 21 separate R&D reports and consolidates it into a single report, thereby eliminating redundancy and insuring data consistency.

MARDIS DOCUMENTATION DATA BASE

The following documents are being used to evaluate the MARDIS software sytem:

- Source Code Listings
- General Functional Specification Requirements
- Detailed Functional Specification Requirements
- Project Master Plan
- System Documentation (Volumes I through VI)
- Software Change Requests
- REMARCS Manpower Data
- MARDIS Conversion Plan/Report

The MARDIS system has 28 COBOL programs, consisting of approximately 54,000 lines of code. Because of the unique requirement for high portability which the U.S. Army Computer Systems Command must satisfy in each of its delivered systems, a special preprocessor is used to accomodate the COBOL dialects implemented on the IBM, CDC and UNIVAC computers used at various Army installations.

4.1.2 ISDS OVERVIEW

The Integrated Software Development System (ISDS) is an evolving collection of software tools and aids which has been developed under an independent research and development project at GE. ISDS consists of several subsystems which support the various personnel and processes involved in the software developments. The subsystems are:

- Computerized Interactive Charting System (CINCH)
  - Assists in interactive development of graphic design material.
- Chart Analysis Subsystem
  - Performs various analyses on design material such as standards enforcement, path flow analysis and complexity measure calculation.
- Program Design Language Processor/Analysis Subsystem
  - Accepts, formats, and analyses a program design language. Analyses include calculation of a complexity measure and creation of a hierarchy chart.
- Programming Language Processors/Parser Subsystem
  - Includes structured language preprocessor and generalized parser. Currently parses FORTRAN, JOVIAL, and PASCAL.

The ISDS project is directed at developing practical methods for improving the software development process throughout all phases of development from requirements analysis to maintenance. In particular, it is concerned with tools which aid in reducing problems associated with the high cost of software development, satisfying customer requirements, meeting contract shedules, and generating adequate documentation.

ISDS DOCUMENTATION DATA BASE

The following documents where used to evaluate ISDS:

Software Development and Implementation Aids IR&D Project Final Report for 1974, GE TIS 75CIS01, P. Richards and P.Chang, July 1975.

Software Development and Implementation Aids IR&D Project Final Report for 1975, GE TIS 76CIS01, P. Richards and P. Chang, January 1976.

Locialization of Variables: A Measure of Program Complexity, GE TIS 76CIS07, P. Richards and P. Chang, December 1976.

Enhancements to the Integrated Software Development System (GE/ISDS), GE TIS 76CIS04, C. Lopez de Nava and W. Neff, December 1976.

Developing Design Aids for an Integrated Software Development System, Proceeding of 1977 Computers in Aerospace Conference, P. Richards, December 1977.

"The Integrated Software Development System - ISDS Users Manual for RSC-11D and RSX-11M", GE Working Paper, de Nava, C., September 1978.

"Computerized Interactive Charting System - Program Specifications", GE Working Paper WP76SEL03, de Nava, C., October 1976.

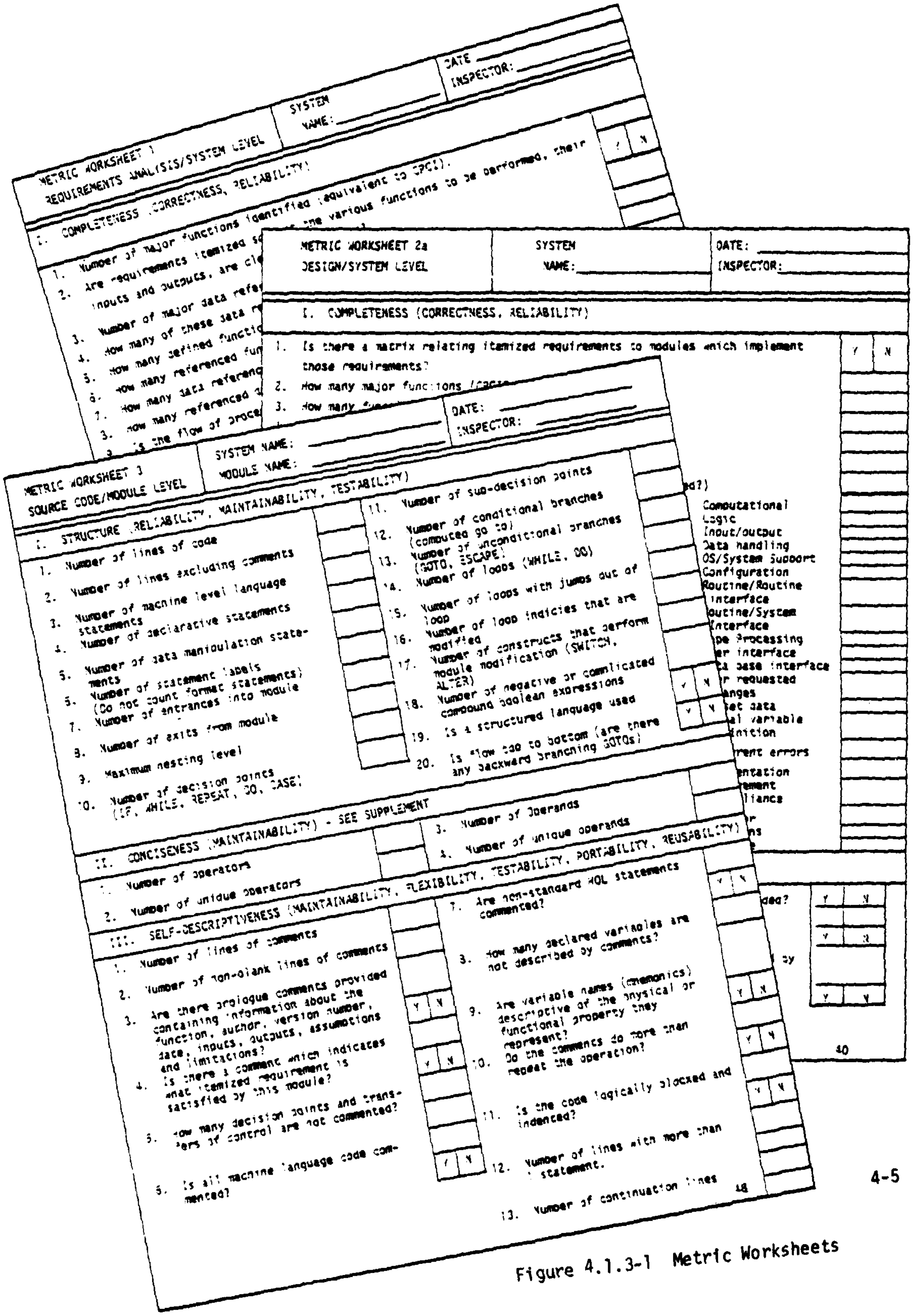
These documents, except for the last two, are technical reports describing IR&D efforts. Contained within each report is a statement of a problem to be solved or a requirement for an additional capability (tool) for the GE/ISDS, a statement of the design approach, and the results of the project. Each document therefore contains the progressive information usually provided during a large scale development effort. They will be utilized in that manner for application of the metrics. The last two documents are more typical of documents found in normal system developments, a users manual and a program specification document.

4.1.3 APPLICATION OF THE METRICS

The format with which the metrics are presented in Appendix B is conducive to illustrating their relationship with respect to the criteria and factors and their progressive application during the phases of development. It is not conducive to actually applying the metrics, i.e., taking the measurements from the product available during development. The purpose of the second volume of this report, the Software Quality Measurement Manual, is to describe the procedures for applying the metrics during a development. The tool which was developed to facilitate application of the metrics in a formal manner are worksheets. The worksheets are described in the second volume of this report. A sample is shown in Figure 4.1.3-1. The worksheets facilitate the manual collection of the raw data used to calculate the metrics. They are organized by phase, by system or module level measurements, and to provide a systematic, organized view of the product being inspected.

To demonstrate the thoroughness or coverage provided by the worksheets, Figure 4.1.3-2 contains a very simple example of the elements of a module that were examined or counted using the source code worksheet (Metric Worksheet 3). The individual elements are underlined if a question or count on the worksheet caused the inspector to look at the particular element. Note the completeness of the coverage. These worksheets are oriented toward the manual applications of the measurements. If tools exist in a particular environment which allows some of the metrics to be taken automatically then the worksheets can simply serve as a bookkeeping form for those particular measurements.

The worksheets presented in the manual represent the final form derived during this study. A preliminary form was used to take the measurements from the ISDS and MARDIS products.



METRIC WORKSHEET 1
REQUIREMENTS ANALYSIS/SYSTEM LEVEL

SYSTEM NAME: ______

DATE ______
INSPECTOR: ______

I. COMPLETENESS (CORRECTNESS, RELIABILITY)

Y N

1. Number of major functions identified (equivalent to CPCI).
2. Are requirements itemized so ... the various functions to be performed, their inputs and outputs, are cle...
3. Number of major data refe...
4. How many of these data r...
5. How many defined functio...
6. How many referenced fun...
7. How many data referenc...
8. How many referenced d...
9. Is the flow of proce...

METRIC WORKSHEET 2a
DESIGN/SYSTEM LEVEL

SYSTEM NAME: ______

DATE: ______
INSPECTOR: ______

I. COMPLETENESS (CORRECTNESS, RELIABILITY)

1. Is there a matrix relating itemized requirements to modules which implement those requirements? Y N
2. How many major functions (CPCIs ...
3. How many fun...

...ed?)

Computational
Logic
Input/output
Data handling
OS/System Support
Configuration
Routine/Routine interface
Routine/System interface
...pe Processing
...er interface
...a base interface
...r requested
...anges
...et data
...al variable
...inition
...rent errors
...entation
...ement
...liance
...r
...ns

...ded? Y N

Y N

...by

Y N

40

METRIC WORKSHEET 3
SOURCE CODE/MODULE LEVEL

SYSTEM NAME: ______
MODULE NAME: ______

DATE: ______
INSPECTOR: ______

I. STRUCTURE (RELIABILITY, MAINTAINABILITY, TESTABILITY)

1. Number of lines of code
2. Number of lines excluding comments
3. Number of machine level language statements
4. Number of declarative statements
5. Number of data manipulation statements
6. Number of statement labels (Do not count format statements)
7. Number of entrances into module
8. Number of exits from module
9. Maximum nesting level
10. Number of decision points (IF, WHILE, REPEAT, DO, CASE)
11. Number of sub-decision points
12. Number of conditional branches (computed go to)
13. Number of unconditional branches (GOTO, ESCAPE)
14. Number of loops (WHILE, DO)
15. Number of loops with jumps out of loop
16. Number of loop indicies that are modified
17. Number of constructs that perform module modification (SWITCH, ALTER)
18. Number of negative or complicated compound boolean expressions
19. Is a structured language used Y N
20. Is flow top to bottom (are there any backward branching GOTOs) Y N

II. CONCISENESS (MAINTAINABILITY) - SEE SUPPLEMENT

1. Number of operators
2. Number of unique operators
3. Number of operands
4. Number of unique operands

III. SELF-DESCRIPTIVENESS (MAINTAINABILITY, FLEXIBILITY, TESTABILITY, PORTABILITY, REUSABILITY)

1. Number of lines of comments
2. Number of non-blank lines of comments
3. Are there prologue comments provided containing information about the function, author, version number, date, inputs, outputs, assumptions and limitations? Y N
4. Is there a comment which indicates what itemized requirement is satisfied by this module? Y N
5. How many decision points and transfers of control are not commented?
6. Is all machine language code commented? Y N
7. Are non-standard HOL statements commented? Y N
8. How many declared variables are not described by comments?
9. Are variable names (mnemonics) descriptive of the physical or functional property they represent? Y N
10. Do the comments do more than repeat the operation? Y N
11. Is the code logically blocked and indented? Y N
12. Number of lines with more than 1 statement.
13. Number of continuation lines

48

Figure 4.1.3-1 Metric Worksheets

]

```
      SUBROUTINE TABSCH(NSYM,I,NFLAG)
C
      COMMON/POLTAB/NPT
      DIMENSION NPT(2ØØ)
C
C        SEARCH THRU DATA BLOCK NPT FOR THE LENGTH UNIT
C        OF THE SYMBOL
C

      NFLAG = Ø
      MAXSYM = 2ØØ
C
      DO 2ØØ J = 1,MAXSYM
C
      IF (NPT(J) .EQ. Ø) GOTO 3ØØ
      IF (NSYM .EQ. NPT(J)) GOTO 1ØØ
C
      J= J+2*NPT(J+1)+4
      GOTO 2ØØ
C
 1ØØ  I = J
      NFLAG = 1
      J = MAXSYM
C
 2ØØ  CONTINUE

 3ØØ  RETURN
      END
```

Figure 4.1.3-2 Worksheet Coverage

4.2 VALIDATION APPROACH

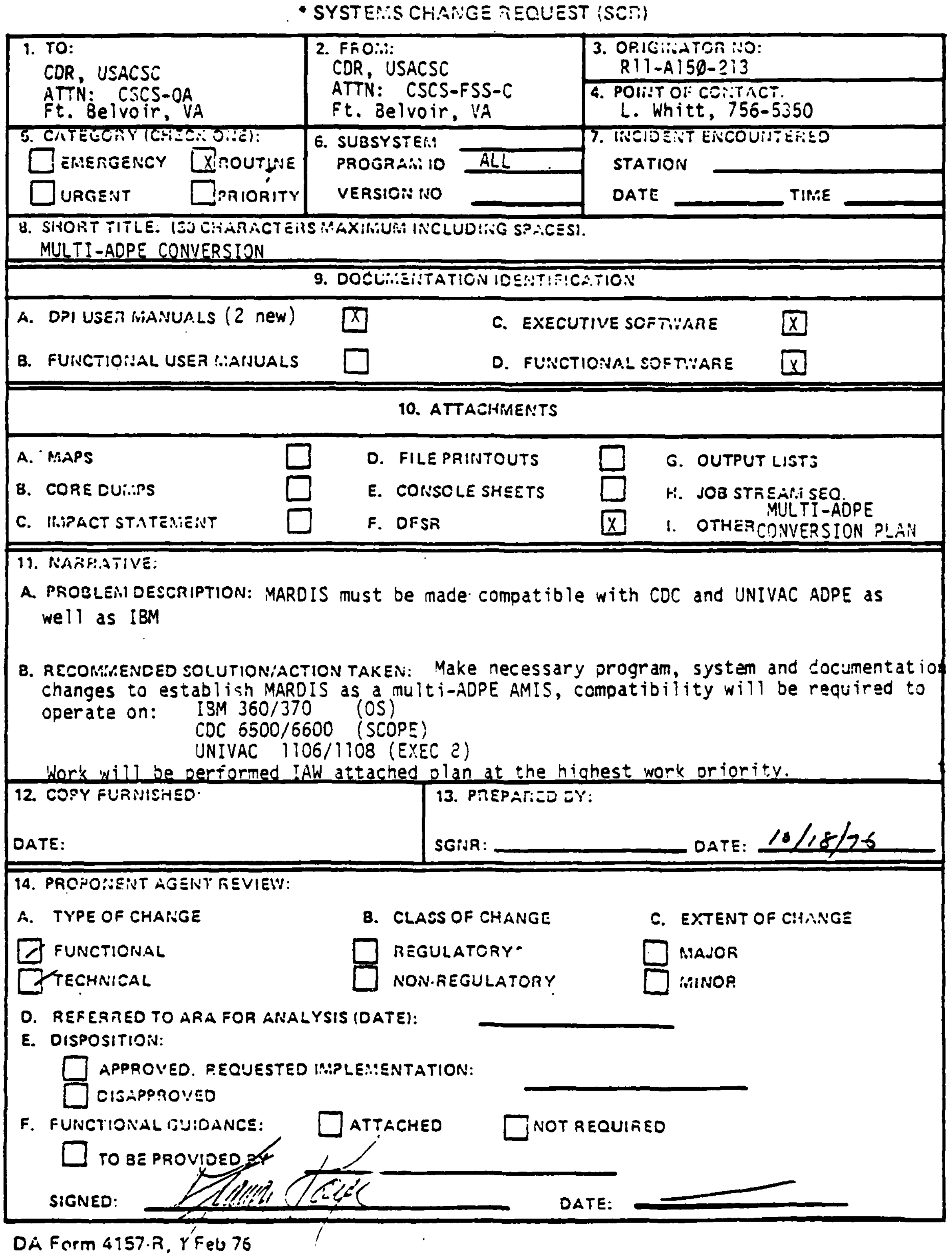
The validation approach was basically the approach described in RADC-TR-77-369 and in paragraph 1.3 of this report. This approach was augmented with some additional analyses made possible by the historical data available. These analyses and some assumptions that were necessarily made due to the historical data available will be described in the subsequent paragraphs.

4.2.1 MARDIS HISTORICAL DATA

The historical data provided about the MARDIS system development and operation consisted of (1) a Final Report documented after the effort to make the system compatible with the three computer evnironments, (2) System Change Requests (SCRs), and (3) a resource accounting (REMARCS) system listing.

Seventy-six (76) SCRs were provided. An example is in Figure 4.2.1-1. Typically an SCR is documented when a problem or enhancement has been identified. The problem is identified, a solution recommended, and a priority assigned for completion of the necessary modification. The resource accounting system maintains person hour expenditures against the SCRs. Over 50 man years of effort were recorded on the REMARCS listings provided.

Our initial intention was to use this data to provide the quality rating for MARDIS. However, because (1) we found little overlap between the SCR's and the REMARCS data, (2) the SCR's were not very explanatory, and (3) only 2 SCRs seem to cover the entire conversion effort, these data sources were only used as a gross indication of the effort and as insight into the problems that were encountered. Instead, we utilized the number of changes that were made to the source code and the problems identified in the final report as more detailed indicators of the portability of the system. The changes were identified by the multi-line entries in the source code relating to the different computer environments. Our



• SYSTEMS CHANGE REQUEST (SCR)

1. TO:
CDR, USACSC
ATTN: CSCS-OA
Ft. Belvoir, VA

2. FROM:
CDR, USACSC
ATTN: CSCS-FSS-C
Ft. Belvoir, VA

3. ORIGINATOR NO:
R11-A15Ø-213

4. POINT OF CONTACT.
L. Whitt, 756-5350

5. CATEGORY (CHECK ONE):
□ EMERGENCY ☒ ROUTINE
□ URGENT □ PRIORITY

6. SUBSYSTEM
PROGRAM ID ALL
VERSION NO

7. INCIDENT ENCOUNTERED
STATION
DATE TIME

8. SHORT TITLE: (30 CHARACTERS MAXIMUM INCLUDING SPACES).
MULTI-ADPE CONVERSION

9. DOCUMENTATION IDENTIFICATION

A. DPI USER MANUALS (2 new) ☒
B. FUNCTIONAL USER MANUALS □
C. EXECUTIVE SOFTWARE ☒
D. FUNCTIONAL SOFTWARE ☒

10. ATTACHMENTS

A. MAPS □
B. CORE DUMPS □
C. IMPACT STATEMENT □
D. FILE PRINTOUTS □
E. CONSOLE SHEETS □
F. DFSR ☒
G. OUTPUT LISTS
H. JOB STREAM SEQ.
I. OTHER MULTI-ADPE CONVERSION PLAN

11. NARRATIVE:

A. PROBLEM DESCRIPTION: MARDIS must be made compatible with CDC and UNIVAC ADPE as well as IBM

B. RECOMMENDED SOLUTION/ACTION TAKEN: Make necessary program, system and documentation changes to establish MARDIS as a multi-ADPE AMIS, compatibility will be required to operate on:
IBM 360/370 (OS)
CDC 6500/6600 (SCOPE)
UNIVAC 1106/1108 (EXEC 2)
Work will be performed IAW attached plan at the highest work priority.

12. COPY FURNISHED·
DATE:

13. PREPARED BY:
SGNR: DATE: 10/18/76

14. PROPONENT AGENT REVIEW:

A. TYPE OF CHANGE
☑ FUNCTIONAL
☑ TECHNICAL

B. CLASS OF CHANGE
□ REGULATORY*
□ NON-REGULATORY

C. EXTENT OF CHANGE
□ MAJOR
□ MINOR

D. REFERRED TO ARA FOR ANALYSIS (DATE):

E. DISPOSITION:
□ APPROVED. REQUESTED IMPLEMENTATION:
□ DISAPPROVED

F. FUNCTIONAL GUIDANCE: □ ATTACHED □ NOT REQUIRED
□ TO BE PROVIDED BY

SIGNED: DATE:

DA Form 4157-R, 1 Feb 76

Figure 4.2.1-1 Example System Change Request

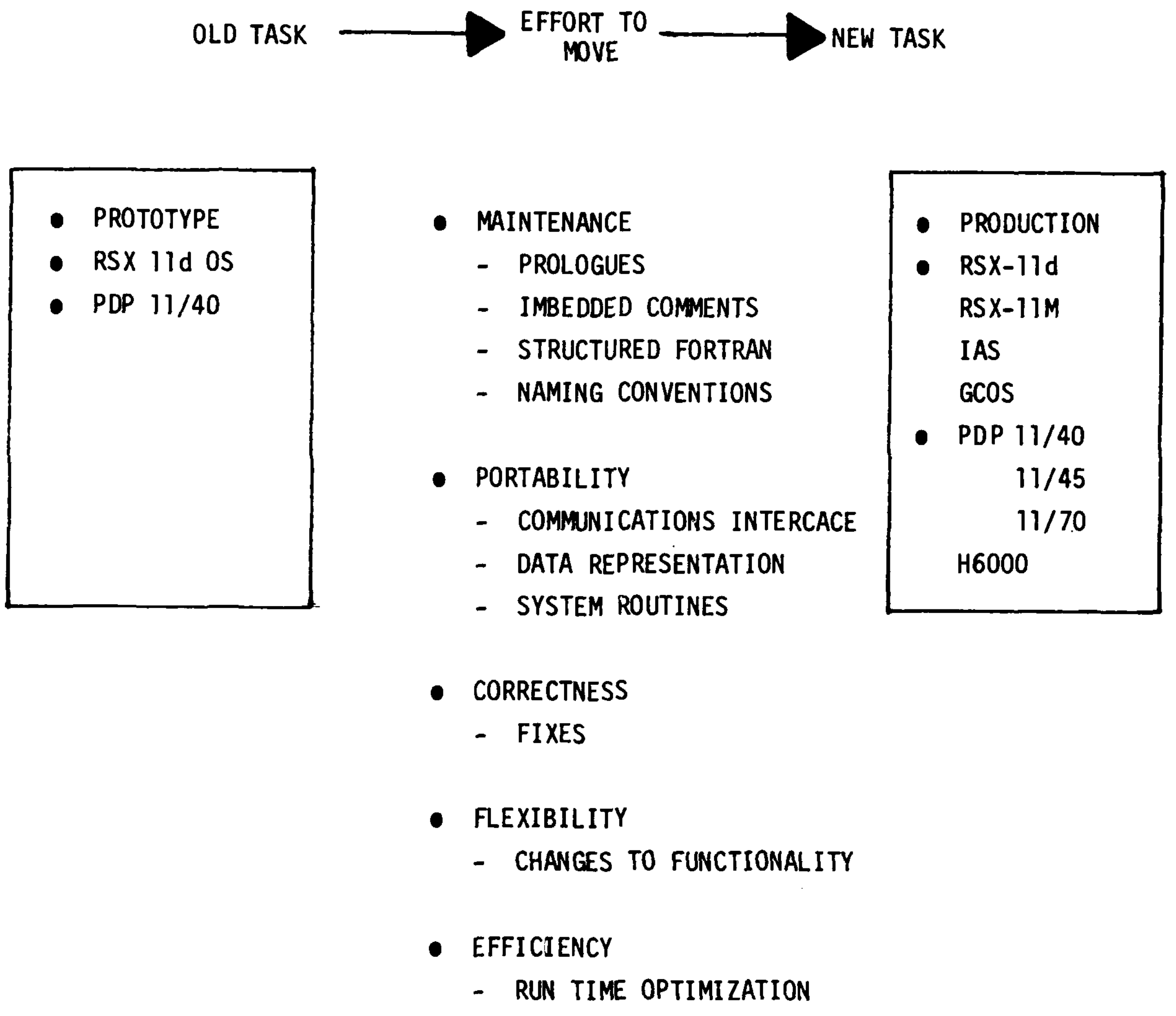
assumption is that the effort to transport the original system is proportional to the number of changes in the source code, This assumption was made basically because the data available did not allow more detailed analysis. Other changes described in the final report but not identified by multi-line entries were analyzed for impact.

The data available supported formal analysis of portability. Less formal assessments were made concerning the maintainability of the software.

4.2.2 ISDS HISTORICAL DATA

The ISDS system was developed as a prototype tool in an R&D environment. As such, a formal software problem reporting system was not in place during its development. In the past year, ISDS has been transferred to a number of GE installations. In transfering the system, considerable effort was made to transition the software from a prototype version to a production tool. This effort during a four month period was captured by use of a data collection form. The form was designed to collect the effort being expended on an ISDS task (a functionally-related group of routines) by quality factor. For example, if changes were being made to enable transportation of a task to a different operating system, the effort to make those changes was recorded against Portability. If changes were made to enhance the Maintainability of the software, such as the addition of standard format prologue comments, use of a structured language, conforming with naming conventions, etc., then effort was recorded against Maintainability. Figure 4.2.2-1 illustrates how we captured the effort required to transport from the prototype version of ISDS on a PDP 11/40 running under the RSX 11d operating system to a number of other environments. The original source code was maintained as well as the new version.

The documentation supporting ISDS is also representative of an R&D or support software environment. The documentation did not represent formal specifications but were instead technical reports describing the R&D project. Involved in transitioning ISDS to a production tool



OLD TASK → EFFORT TO MOVE → NEW TASK

- ● PROTOTYPE
- ● RSX 11d OS
- ● PDP 11/40

- ● MAINTENANCE
  - PROLOGUES
  - IMBEDDED COMMENTS
  - STRUCTURED FORTRAN
  - NAMING CONVENTIONS
- ● PORTABILITY
  - COMMUNICATIONS INTERCACE
  - DATA REPRESENTATION
  - SYSTEM ROUTINES
- ● CORRECTNESS
  - FIXES
- ● FLEXIBILITY
  - CHANGES TO FUNCTIONALITY
- ● EFFICIENCY
  - RUN TIME OPTIMIZATION

- ● PRODUCTION
- ● RSX-11d
RSX-11M
IAS
GCOS
- ● PDP 11/40
11/45
11/70
H6000

Figure 4.2.2-1 ISDS Data Collection

has been the development of documentation which will support its operation and maintenance. The documentation was not available during this study.

4.3 VALIDATION RESULTS

The validation results are presented at three levels corresponding to the three levels of quality assessment analysis described in the Software Quality Measurement Manual (volume II): Inspectors Assessment, Sensitivity Analysis, and use of normalization function.

4.3.1 INSPECTORS ASSESSMENT

In evaluating the MARDIS and ISDS systems, a qualitative assessment of the code was made as part of the investigation. The qualitative assessment (review or audit) would normally be part of a quality assurance program and is enhanced by use of the worksheets and quantitative measurements. The assessment identifies problem areas which should be addressed in subsequent phases of the development.

GENERAL OBSERVATIONS OF MARDIS

Two quality factors analyzed in MARDIS were Portability and Maintainability.

PORTABILITY

Portability is a quality factor which is important to the Army Computer Systems Command because of the operating environment in which it must exist. This operating environment consists of multiple mainframes and operating systems, and the attendant incompatibilities inherent in such an environment. These incompatibilities force an overhead on software development to create portable systems. In the instance of MARDIS, a major redesign of the original system was undertaken to make the system compatible with Honeywell and UNIVAC systems. Had such a requirement been identified early in the lifecycle, the transition would have required much less effort.

The portability of a system written in COBOL is still a significant problem even though government purchased computers used in business applications generally use approved COBOL compilers.

The approach used by the MARDIS team was to redesign the code using a strict ANSI COBOL subset. In the process of developing the new system, they found that there was 16 compiler implementation anamolies Table 4.3.1-1 breaks the anamolous cases into 3 categories: I/O, Semantics of Implementation and Character Set. The I/O category is any situation where the operating system, which allocates system input and output resources, interacts with the COBOL compiler. Semantics of Implementation refers to differences in interpreting the language specification for the compiler. Character Set refers to the available character set of the particular machine. Eleven of the anamolies are semantic, 4 are I/O and 1 related to character set.

One tends to expect some I/O related problems, given the multiple hardware environment. The semantical problems, however, are surprising since a considerable amount of initial development effort had been put into the original COBOL specification to enhance portability. The use of a subset of the ANSI standard set alleviated many of the semantic problems for MARDIS. However, the use of "multiple-line code (duplicate statements, each statement or group of statements targeted for a specific machine) was especially necessary for I/O related code.

For example, the program PIOAYE, which updates and loads tables used by other MARDIS programs, has multi-line code in the following areas:

- INPUT - OUTPUT SECTION
- FILE CONTROL
- FILE SECTION
- FILE DESCRIPTIONS (FD)
- requests for CURRENT - DATE
- carriage control areas
- error return codes from SORT (IBM)

Table 4.3.1-1 Compiler Implementation Anamolies

CASE	I/O	SEMANTICS OF IMPLEMENTATION	CHARACTER SET
1. SYSTEM DATES SYSTEM		X	
2. TABLE INITIALIZATION		X	
3. GOTO DEPENDING ON		X	
4. REDEFINES		X	
5. COLON			X
6. FILES IN SUB-PROGRAMS		X	
7. MOVE		X	
8. RANDOMLY ACCESSED FILES		X	
9. INDEXED BY		X	
10. ASSIGN TO	X		
11. SUBSCRIPTS OUT OF RANGE		X	
12. LINAGE CLAUSE	X		
13. SPECIAL-NAMES	X		
14. LENGTH OF PRINT LINE	X		
15. PARAGRAPH NAME		X	
16. DISPLAY VERB		X	

The program P40HAUE, which selects and formats records for the Cost Reduction report, has multi-line code in the following areas:

- CONFIGURATION SECTION
- carriage control areas

It is significant that no multi-line code appeared in the P40HAUE LINKAGE SECTION. This tends to indicate that the areas which impacted the Portability of MARDIS most were those where the compiled code had to interact with the operating system, i.e., in those areas which dealt with I/O.

Thus the design strategy utilized by the MARDIS conversion team is an effective one for controlling Portability - restrict the language used to a subset common to the target machines (necessarily slightly lower in level than any single implementation language) and, in those instances where the compiled code must interact with the operating system, control the code through the use of a pre-processor. Had the standards and conventions in effect during the initial development of MARDIS imposed these restrictions on the developers the conversion effort would have been significantly less.

MAINTAINABILITY

The Maintainability of MARDIS is impacted by the size of the sytem (54,000 lines of code), the lack of modularity, the lack of comments and the evidence of multiple authors. The software criteria which relate to these problems are conciseness, modularity and consistency.

The size of the system makes it less concise. Halstead's measure is used as a metric for this criteria. The combination of the natural verbosity of COBOL and the fact that quite a lot of information about the system is stored in the program, for example as tables, contributes to the relatively larger sizes of COBOL programs. Since the language and application militate against the conciseness of MARDIS, there is fixed "overhead" impact on the system. This must be controlled as much as possible as the system evolves during the lifecycle.

It is difficult to implement modular systems in COBOL because of the difficulty of coding subprograms. This results in larger programs than is generally convenient to read or write. This impact on ease of scanning or reading a program directly effects Maintainability. In order to increase modularity, COBOL programmers tend to "localize" code, so that single functions have all their statements isolated in a paragraph or section. However, this is left to the discretion of the programmer and is not a "natural" attribute of the language. In the case of MARDIS, the average program length exceeded 2,000 lines of code.

The lack of adequate comments in MARDIS makes it difficult to identify the function of particular groups of statements. Generally, COBOL is said to be a "self-descriptive" language, which, relative to many languages, is true. The code, however, only tells us what is happening, not why it is happening. This "why" aspect can be very helpful to the maintenance programmer, since it can help him to identify the function and avoid side effects in his modifications to the code.

In reading MARDIS code it becomes immediately apparent that there were multiple authors involved in its development. On the face of it, this is obvious given the size of the undertaking. Beneath the surface, however, is the realization that a uniform development methodology with adherence to uniform standards and conventions was not used. Uniformity and standardization of coding practice, for example, in indentation or naming of data, is an important aid to establishing Maintainability. This does away with the need to learn many different personal styles of coding and can lead to uniform product quality.

In general, the MARDIS doucmentation provided a good overview of the MARDIS system, its purpose and its operation. However, the documentation did little to support its maintenance. Maintenance-oriented documentation requires a greater level of detail in its description of design and implementation strategies and description of the internals of the system.

GENERAL OBSERVATIONS OF ISDS

The initial version of the ISDS code reflected its development environment, an R&D or prototype development. In such a development environment, more concern is shown in the algorithm development or technique than in the user interface or operability. The initial concern was to build a prototype quickly and utilize the prototype to evaluate the effectiveness of the tool in support of a software development

Most of the personnel involved in the developr ıc of ISDS were aware of and practiced modern programming practices. In almost all cases, structured programming techniques and a structured FORTRAN pre-processor were used. However, there were not coding standards and conventions or enforcement techniques in place in the initial development and the result is that different styles or approaches to modern programming practices are evident. While this is a step above "unstructured" techniques, the lack of consistency has a negative impact on the maintainability of the system, i.e., uncontrolled modern programming practices are not much better than traditional techniques.

The conversion of ISDS to a production environment required considerable effort but has resulted in a much higher quality product. The changes made to the system, illustrated in Figure 4.2.2-1, indicates what specific attributes of the software where enhanced.

The documentation reflected the code. The interest was in the algorithms and functionality of the software. Little concern was shown for operability or methodology of use. The transition to a production tool is attaking that problem.

The modularity of ISDS was excellent. The average size for a module was less than 100 lines of code. This attribute alone had a significant effect on its transportation to other environments and its transition to a production tool.

4.3.2 SENSITIVITY ANALYSIS

The sensitivity analyses proved to be a very effective quality assessment technique. The analyses possible with the quantitative data available from applying the metrics provides an immediately useful quality assurance technique.

MARDIS SENSITIVITY ANALYSIS

Table 4.3.2-1 provides a subset of the statistics of the MARDIS system evaluated. Note the large size of the programs and the large number of branches (most of which are PERFORMS). The profiles provided are interesting from the standpoint that two programs were very large (over 10,000 lines of code) and represent maintenance difficulties just from their size. Also, 27% of the programs contained most of the changes which had to be made in transporting the system (greater than 10% of the code in each of those programs had to be changed). These are the programs which consumed most of the effort. This illustrates a benefit of the metric analyses. These programs are identified and can be emphasized in planning for a conversion effort.

There were very few comments. Only three percent of the lines of code were comments and in fact one-third of the programs contained no comments at all except for a standard ten line comment at the beginning about the multi-computer version implementation.

These type of statistics are not only valuable for quality control but statistics on language construct usage contribute to new standards and conventions and avoidance of future problems [ALJM 79].

Table 4.3.2-1 MARDIS Statistics

PROGRAM SIZE (LINES OF COBOL SOURCE CODE)	TOTAL: 54630			AVERAGE: 2023		NUMBER OF PROGRAMS: 27	
SIZE PROFILE	NUMBER OF MODULES	4	7	11	3	0	2
	SIZE RANGE	0-500	500-1000	1000-2000	2000-5000	5000-10000	> 10000
COMMENTS	AVERAGE: 3% OF LINES OF CODE						
NUMBER OF BRANCHES	AVERAGE: 52						
MACHINE DEPENDENT CODE	AVERAGE: 2.5% OF LINES OF CODE						
MACHINE DEPENDENT CODE PROFILE:	PERCENT OF NUMBER OF PROGRAMS	73%	30%	0%	7%		
	PERCENT OF LINES OF CODE CHANGED	0-10%	10-50%	50-75%	75-100%		

ISDS SENSITIVITY ANALYSIS

Table 4.3.2-2 and 4.3.2-3 provide a subset of the ISDS statistics evaluated. Note the high degree of modularity evidenced by the statistics and the high percentage of comments. In evaluating these statistics at a task level (subsystem level) considerable variance was realized in percent of comments and average number of branches. We were able to establish significant correlation between those statistics and the effort required to transport and enhance the various tasks.

As a more detailed analysis, the metrics related to Maintainability and Portability were calculated for a subset of the modules of the new version of ISDS and compared to the old version. Figure 4.3.2-1 provides some examples. The acronyms are indexes into the metric table in Appendix B.

These values were then compared to a relative indicator of the effort required to transport and enhance the maintainability of ISDS. The relative indicator represents the percent of the total effort to enhance the maintainability or transport the system. Table 4.3.2-4 provides some examples of this analyses. Our analysis of this data was aimed at determining whether the difference between the metric score for the initial version and the new version correlated with the effort required to produce the new version from the old version. The following metrics demonstrated significant correlation (correlation coefficients better than .75):

SI.4	Coding Simplicity Measure
SD.1	Quantity of Comments
SD.2	Effectiveness of Comments Measure
SD.3	Descriptiveness of Language Measure
MO.2	Modular Implementation Measure
SS.1	System Software Independence Measure
MI.1	Machine Independence Measure

A surprising result was that the complexity measure did not improve generally between versions. The reason was because we were using a modification of McCabe's measure and there is no penalty for unconditional branches. Therefore in cases where we went from an unstructured module implementation to a structure implementation, and replaced GOTO's with structured constructs. In these situations, the number of paths may have increased or at least stayed the same, even though the structure was more simple or easier to understand. This attribute was reflected in the Code Simplicity Measure.

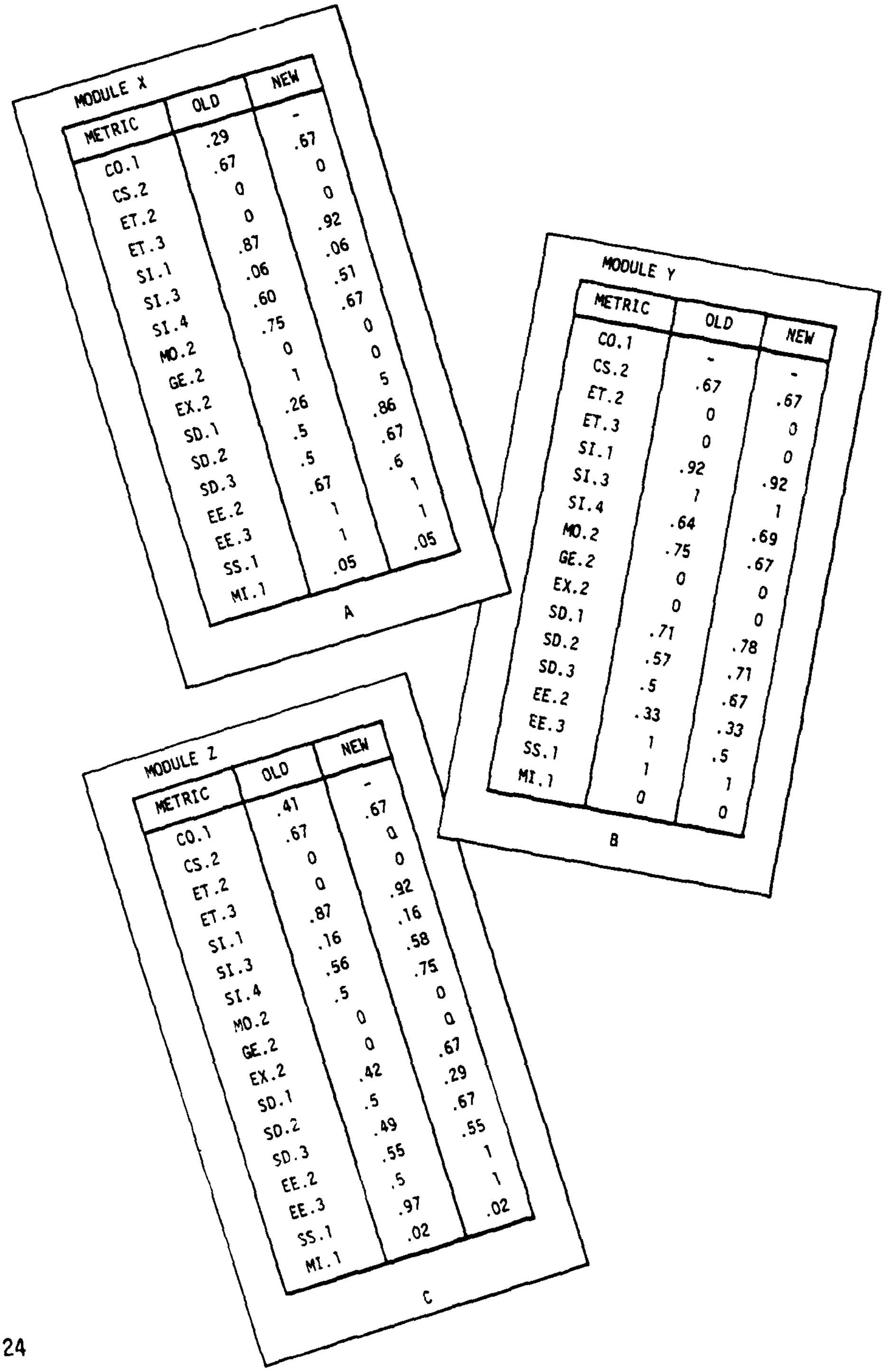
These types of analyses, using the quantitative technical statistics provided by the application of the metrics, can be a very beneficial aid to a quality assurance person. The identification of how certain measures vary in a system gives insight into the adherence to standards, what characteristics need to be controlled by new standards, and which modules vary from the average significantly and should be evaluated further.

Table 4.3.2-2 ISDS Version Comparison Statistics

PROGRAM SIZE (LINES OF FORTRAN SOURCE CODE)	TOTAL OLD VERSION: 18040 NEW VERSION: 27600		AVERAGE: 82 97	NO. OF PROGRAMS: 219 284		
SIZE PROFILE (BASED ON SAMPLE)	NUMBER OF MODULES OLD VERSION NEW VERSION	34% 10%	34% 57%	27% 28%	5% 5%	
	SIZE RANGE	0-50	50-100	100-200	200-500	7500
COMMENTS	AVERAGE: OLD VERSION: 35% NEW VERSION: 62%					
NUMBER OF BRANCHES	AVERAGE: OLD VERSION: 1.65 NEW VERSION: .57					
MACHINE DEPENDENT CODE	AVERAGE: 4%					

Table 4.3.2-3 ISDS Initial Version Statistics

	AVERAGE	STANDARD DEVIATION
TOTAL LINES OF CODE (LOC)	85	54
LOC WITHOUT COMMENTS	52	43
# OF DECISIONS (IF, DO WHILE, ETC)	7.8	7.5
# OF UNCONDITIONAL BRANCHES (GOTOS)	1.6	4.2
# OF EXITS/ ENTRANCES	2.3	.9
# OF LOOPS	2.5	3.2
# OF STATEMENT LABELS	1.6	2.8
# OPERATORS	47.8	35.5
# OPERANDS	59.3	45.8
% COMMENTS	34.	19.
# OF INPUT/OUTPUT STATEMENTS	2.8	4.
# CALLS TO OTHER MODULES	4.5	10.5
# OF SYSTEM SOFTWARE REFERENCES	.4	.9
# OF LOCAL VARIABLES	8.3	6.5
# OF GLOBAL VARIABLES	3.6	3.9



MODULE X

METRIC	OLD	NEW
CO.1	.29	-
CS.2	.67	.67
ET.2	0	0
ET.3	0	0
SI.1	.87	.92
SI.3	.06	.06
SI.4	.60	.51
MO.2	.75	.67
GE.2	0	0
EX.2	1	0
SD.1	.26	5
SD.2	.5	.86
SD.3	.5	.67
EE.2	.67	.6
EE.3	1	1
SS.1	1	1
MI.1	.05	.05

A

MODULE Y

METRIC	OLD	NEW
CO.1	-	-
CS.2	.67	.67
ET.2	0	0
ET.3	0	0
SI.1	.92	.92
SI.3	1	1
SI.4	.64	.69
MO.2	.75	.67
GE.2	0	0
EX.2	0	0
SD.1	.71	.78
SD.2	.57	.71
SD.3	.5	.67
EE.2	.33	.33
EE.3	1	.5
SS.1	1	1
MI.1	0	0

B

MODULE Z

METRIC	OLD	NEW
CO.1	.41	-
CS.2	.67	.67
ET.2	0	0
ET.3	0	0
SI.1	.87	.92
SI.3	.16	.16
SI.4	.56	.58
MO.2	.5	.75
GE.2	0	0
EX.2	0	0
SD.1	.42	.67
SD.2	.5	.29
SD.3	.49	.67
EE.2	.55	.55
EE.3	.5	1
SS.1	.97	1
MI.1	.02	.02

C

Figure 4.3.2-1 Metric Score Comparisons

Table 4.3.2-4 ISDS Sensitivity Analysis

	ROUTINE A				ROUTINE B				ROUTINE C		
	OLD	NEW	EFFORT		OLD	NEW	EFFORT		OLD	NEW	EFFORT
MAINT											
COMPLEXITY	.33	.25	.20		0	0	.09		.25	.25	.13
CODE SIMP	.71	.76			.82	.88			.71	.80	
COMMENTS	.22	.59			.70	.78			.50	.73	
EFF. OF COMMENTS	0	.6			.8	1.			.4	.6	
PORT											
S/W SYS. INDEP.	.75	1.	.05		1	1	.08		.75	.75	.18
MACH INDEP.	0	.5			.5	.75			.5	.75	

Table 4.3.2-4 ISDS Sensitivity Analysis (Continued)

	ROUTINE D				ROUTINE E				ROUTINE F		
	OLD	NEW	EFFORT		OLD	NEW	EFFORT		OLD	NEW	EFFORT
MAINT											
COMPLEXITY	.07	.06	.09		10	10	.13		-	-	.30
CODE SIMP	.64	.74			.81	.876			.77	.93	
COMMENTS	.26	.49			.4	.6			.27	.60	
EFF. OF COMMENTS	.4	1			.4	.6			.20	1.0	
PORT											
S/W SYS INDEP.	1	1	.08		.75	1	.06		.625	.875	.39
MACH INDEP	.25	.5			.25	.5			.588	.81	

Table 4.3.2-4 ISDS Sensitivity Analysis (Continued)

	ROUTINE G				ROUTINE H				ROUTINE I		
	OLD	NEW	EFFORT		OLD	NEW	EFFORT		OLD	NEW	EFFORT
MAINT											
COMPLEXITY	-	-	.05		.06	.06	.01		.25	.25	.02
CODE SIMP	.88	.88			.60	.51			.62	.54	
COMMENTS	.32	.47			.26	.50			.51	.73	
EFF. OF COMMENTS	.57	.86			.5	.86			.5	.57	
PORT											
S/W SYS. INDEP	.875	.875	.16		1	1	0		.96	.96	.01
MACH INDEP.	.65	.875			.05	.05			.1	.1	

4.3.3 NORMALIZATION FUNCTION ANALYSIS

The normalization functions established in RADC-TR-77-369 [McCJ77] were modified or further validated based on the additional data sample provided by this effort. In addition, a normalization function for Portability was established. The same process used in RADC-TR-77-369 was used in this study.

Table 4.3.3-1 provides a summary of the metric scores (average score and standard deviation) achieved by MARDIS and ISDS. These scores were calculated as a consistency check to evaluate if the metric scores were reasonable with respect to our experience of applying them to JOVIAL code during the Factors in Software Quality study.

The results of the normalization functiona analysis and derivation are shown in Table 4.3.3-2. The table identifies a multivariate relationship as well as normalization functions calculated for individual metrics. In most cases, the multivariate normalization function would be the preferred relationship to use because greater precision can be achieved with it. The metrics related to Maintainability at implementation were an exception to this statement. Some of the individual metrics showed higher correlation. However, in this case, the sample size or the fact that several dimensions should be examined would still encourage the use of the multivariate relationship. However, in cases where data availability or effort to measure are limited, single metric relationships can be used.

Caution must be taken in using these results. Our sample size, even though we have now applied the metrics to two JOVIAL command and control systems (∿ 40,000 lines of code), and COBOL financial management information system (∿ 50,000 lines of code) and a FORTRAN software support system (∿ 20,000 lines of code), is still small to place too much confidence in the results. It is significant, however, that relationships do exist and our intuitions in establishing the measurements have statistical reinforcement. As shown in Table 4.3.3-3, the measurements that exhibit correlation to Portability and Maintainability, the quality factors emphasized during this study, are logical. The metrics that did not exhibit significant correlations are shown in Table 4.3.3-4. An explanation or description of the action taken is provided.

Table 4.3.3-1 Metric Scores

METRIC	AVERAGE SCORE	SATANDARD DEVIATION
CO.1	.12	.14
CS.2	.68	.42
ET.2	.02	.07
ET.3	.07	.2
SI.1	.87	.09
SI.3	.23	.25
SI.4	.57	.07
MO.2	.71	.18
GE.2	.35	.47
EX.2	.07	.26
SD.1	.35	.16
SD.2	.40	.19
SD.3	.57	.1
EE.2	.50	.16
EE.3	.85	.23
SS.1	.01	.01
MI.1	.21	.49

Table 4.3.3-2 Normalization Functions

QUALITY FACTOR/NORMALIZATION FUNCTIONS		STANDARD ERROR	CORRELATION COEFFICIENT
RELIABILITY (DESIGN)			
Multivariate Function	.18 M(ET.1) + .19 M(SI.3)	.17	.87
Individual Functions	.34 M(ET.1)	.18	.82
	.34 M(SI.3)	.16	.85
RELIABILITY (IMPLEMENTATION)			
Multivariate Function	.48 M(ET.1) + .14 M(SI.1)	.33	.85
Individual Functions	.57 M(ET.1)	.31	.83
	.58 M(SI.1)	.31	.78
	.53 M(SI.3)	.32	.78
	.53 M(SI.4)	.34	.77
MAINTAINABILITY (DESIGN)			
Individual Functions	.67 M(SI.3)	.28	.88
	.53 M(SI.1)	.27	.83
MAINTAINABILITY (IMPLEMENTATION)			
Multivariate Function	.61 M(SI.3) + .14 M(MO.2) + .33 M(SD.2) -.2	.06	.78
Individual Functions	2.1 M(SI.3)	.185	.89
	.71 M(SD.2)	.29	.74
	.6 M(SD.3)	.23	.84
	.48 M(SI.1)	.15	.91
	.43 M(SI.4)	.17	.89

Table 4.3.3-2 Normalization Functions (Continued)

QUALITY FACTOR/NORMALIZATION FUNCTIONS		STANDARD ERROR	CORRELATION COEFFICIENT
FLEXIBILITY IMPLEMENTATION)			
Multivariate Function	.22 M(MO.2) + .44 M(GE.2) +.09 M(SD.3)	.11	.98
	.6 M(MO.2)	.12	.96
	.72 M(GE.2)	.15	.93
	.59 M(SD.2)	.16	.95
	.56 M(SD.3)	.14	.96
PORTABILITY (IMPLEMENTATION)			
Multivariate Function	.19 M(SD.1) + .76 M(SD.2) + .25 M(SD.3) + .64 M(MI.1)	.05	.93
Individual Functions	1.07 M(SI.1)	.28	.90
	1.1 M(MI.1)	.33	.90
	1.5 M(SD.2)	.39	.86

Table 4.3.3-3 Results of Normalization Function Analysis
(Individual Metrics Which Exhibited
Correlation to Quality Factors)

PORTABILITY	
SD.1	Quantity of Comments
SD.2	Effectiveness of Comments
SD.3	Descriptiveness of Language
MO.2	Modular Implementation
MI.1	Machine Independence
MAINTAINABILITY	
SI.1	Design Structure Measure
SI.3	Complexity
SI.4	Coding Simplicity
MO.2	Modular Implementation
SD.2	Effectiveness of Comments

Table 4.3.3-4 Results of Normalization Function Analysis (Individual Metrics Which Did Not Exhibit Significant Correlation)

FACTOR/METRIC		EXPLANATION
PORTABILITY		
MO.1	Degree of Independence (Myer's)	System level metric - not calculated
SS.1	Software System Independence	System dependencies were for the most part alleviated by the use of multi-line entries in MARDIS code. Further evaluation required.
MAINTAINABILITY		
CS.1	Procedure Consistency	System level metric - considered anomaly detecting metrics
CS.2	Data Consistency	System level metric - considered anomaly detecting metrics
SI.2	Structured Programming	Little variation found - dropped from candidate metrics
MO.1	Degree of Independence (Myer's)	System level metric - not calculated
SD.1	Quantity of Comments	The percentage of comments alone did not show significant correlation - considered anomaly detecting metric

Table 4.3.3-4 Results of Normalization Function Analysis (Continued)

FACTOR/METRIC		EXPLANATION
SD.3	Descriptiveness of Language	Did not exhibit much variation within system
CO.1	Conciseness (Halstead's length)	The comparison of calculated length with observed length varied greatly

APPENDIX A

PRODUCTS PRODUCED IN TYPICAL ARMY SOFTWARE DEVELOPMENT

This is a listing of documentation required by the Army for software development according to CSCM 18-1.

SYSTEM DOCUMENTATION REQUIREMENTS
CSCM 18-1, Paragraph 6.1.1.4

A System Overview will be prepared by the developer whenever a multicommand system has two or more subsystems. The manual provides the needed interface between subsystems in a modularly developed ADP system.

SYSTEM DOCUMENTATION REQUIREMENTS
CSCM 18-1, Paragraph 6.1.1.5

A separate six volume manual will be published, using the assigned unique subsystem identification code for each application subsystem and for each executive software subsystem maintaining a baseline as defined in Chapter 1 of this manual and qualifying as a integrated assembly of separate but functionally interrelated programs, routines, procedures, or techniques operating in consonance as an entity in the performance of a predefined functional ADP task. Each of the six volumes is specifically tailored to the various users of the subsystem and is intended to be self-contained. The six volumes of documentation required for the application subsystems and for the executive software subsystems determined to perform predefined functional ADP tasks are described in the following paragraphs.

EXECUTIVE SUMMARY, VOL I
CSCM 18-1, Paragraph 6.1.1.5.1

This volume will contain an overview of the subsystem to include the objectives and general description written in nontechnical language.

USER DOCUMENTATION, VOL II
CSCM 18-1, Paragraph 6.1.1.5.2

This document prescribes the procedures that must be followed for successful utilization of the subsystem. It contains instructions for the general use of the subsystem, preparation of input, audit of output and interface procedures between the functional area and the data automation activity. The document will be presented in one of the two following formats: Functional User Documentation or Users Procedures, Volume II. Users Procedures, Volume II, is intended for those subsystems where the ADP developer is also the proponent.

OPERATIONS AND SCHEDULING, VOL III
CSCM 18-1, Paragraph 6.1.1.5.3

This volume will contain instructions necessary to schedule the subsystem, run the computer, produce outnut products, and distribute the results.

REFERENCE MATERIAL, VOL IV
CSCM 18-1, Paragraph 6.1.1.5.4

This volume will contain all of the material which preceded the technical design and analysis of the subsystem. This volume will not be distributed below the agency maintaining the subsystem.

GENERAL SYSTEM ANALYSIS DOCUMENTATION, VOL V
CSCM 18-1, Paragraph 6.1.1.5.5

This volume will contain all of the material used or developed during the technical design and analysis of the subsystem. This volume will not be distributed below the agency maintaining the subsystem.

SYSTEM PROGRAM DOCUMENTATION, VOL VI
CSCM 18-1, Paragraph 6.1.1.5.6

This volume will contain completed material necessary to understand the purpose and processing of a program used and developed during the programming and testing of the subsystem. This volume will not be distributed below the agency maintaining the subsystem.

EXECUTIVE SOFTWARE AND SOFTWARE DEVELOPMENT TOOLS DOCUMENTATION REQUIREMENTS
CSCM 18-1, Paragraph 6.1.1.6

This portion of the chapter defines the detailed documentation requirements for documenting executive software and development tools: i.e., macro instructions, subroutines, stand-alone utility programs, and utility systems operating primarily in support of functional applications subsystems not maintaining a baseline as defined in the preceding paragraph.

TECHNICAL PAMPHLETS
CSCM 18-1, Paragraph 6.1.1.7

Technical pamphlets are used to provide DPI ADP personnel with information such as language syntax, input sequencing, input coding instructions, and other procedures necessary to utilize USACSC systems, subsystems, and executive software. The use of technical pamphlets, however, does not preclude the requirement for documentation of the software in application subsystem or executive software documentation format as described above. The content of a technical pamphlet will be determined by its proponent. The preparation of a technical pamphlet will be the same; in general, as for the preparation of application subsystem documentation.

APPENDIX B
SOFTWARE QUALITY METRICS

The metrics established in [MCCJ77] have been refined based on the experiences of this research study. The changes are contained in this appendix. The changes are indicated to the far right of the table as follows:

m - A modification has been to the previous metric to make it more generally applicable or to quantify it.
d1 - The metric was deleted because it was too difficult to measure.
d2 - The metric was deleted because it was not generally applicable.
d3 - The metric was deleted because it was redundant with another metric.
a - The metric was added based on further research.

Also indicated in the table that follows is whether the current state of the metric makes it an anomaly-detecting metric or a predictive metric. If a normalization function has not been established for the quality factor the metric corresponds to then it is automatically an anomaly-detecting metric. In cases where a normalization function has been established for a quality factor but the metric is not included it is because the metric did not illustrate sufficient correlation with the operational history. In lieu of inclusion in the normalization function, some metrics are maintained as strictly anomaly-detecting metrics. They are felt to identify or assist in identification of problems which should be and are typically corrected immediately to enhance the quality of the product. An (a) beneath the criterion/subcriterion name identifies an anomaly detecting metric and a (p) identifies a predictive metric. As further research in software metrics continue, more predictive metrics will be identified.

One last indication has been added to the table. Within the boxes that identify during what phase a particular measurement element can be taken, a reference to what Metric Worksheet applies is given. The reference is to the worksheet number such as Worksheet 1 or Worksheet 2a and what section of the worksheet such as Section III or Section VI. The Metric Worksheets are contained in the Software Quality Measurement Manual [Vol. II]. Explanations of the individual measurement elements follow the table in this appendix.

FACTOR(S): CORRECTNESS

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION		
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	
TRACEABILITY (a)	TR. 1 Cross reference relating modules to requirements.			2aI		3III		m
	SYSTEM METRIC VALUE: Same as above line			□		□		
COMPLETENESS (a)	CP.1 COMPLETENESS CHECKLIST:							
	(1) Unambiguous references (input, function, output)	1 I		2bI		2bI		
	(2) All data references defined, computed, or obtained from an external source. (Data references defined/total data references)		1 I		2bI		2bI	m
	(3) All defined functions used. (Defined function used/total functions identified)		1 I		2aI		2aI	m
	(4) All referenced functions defined. (Referenced functions defined/total functions identified)		1 I		2aI		2aI	m
	(5) All conditions and processing defined for each decision point.	1 I		2bI		2bI		
	(6) All defined and referenced calling sequence parameters agree. (Parameters agree/total parameters)				2aI		2aI	m
	(7) All problem reports resolved. (Problem reports resolved/total problem reports)	1 I			2aI		2bI	m
	(8) Design agrees with requirements.			□				dl
	(9) Code agrees with design.					□		dl
	METRIC VALUE = $\frac{\sum_{1}^{7} \text{Score for element } i}{7}$		□		□		□	

C&IS FORM 140 (13 JAN 77)

FACTOR(S): CORRECTNESS, RELIABILITY, MAINTAINABILITY

CRITERION/ SUBCRITERIUN	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
CONSISTENCY/ PROCEDURE CONSISTENCY (a)	CS. 1 PROCEDURE CONSISTENCY MEASURE						
	(1) Standard design representation $\left(1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ modules}}\right)$				2bXI		
	(2) Calling sequence conventions $\left(1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ modules}}\right)$				2bXI		2bXI
	(3) Input /output conventions $\left(1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ modules}}\right)$				2bXI		2bXI
	(4) Error handling conventions $\left(1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ modules}}\right)$				2bXI		2bXI
	METRIC VALUE = $\frac{\text{Sum of scores of applicable elements}}{\#\text{ of applicable elements}}$				□		□
DATA CONSISTENCY (a)	CS. 2 DATA CONSISTENCY MEASURE						
	(1) Standard data usage representation $\left(1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ modules}}\right)$				2bXI		
	(2) Naming conventions $\left(1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ modules}}\right)$				2bXI		2bXI
	(3) Consistent global definitions $\left(1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ modules}}\right)$				2bXI		2bXI
	(4) Unit consistency $\left(1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ modules}}\right)$				□		□ d1
	(5) Data type consistency $\left(1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ modules}}\right)$				□		□ d1
	METRIC VALUE = $\frac{\text{Sum of scores of applicable elements}}{\#\text{ of applicable elements}}$				□		□

FACTOR(S): RELIABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE
ACCURACY (a)	AC. 1 ACCURACY CHECKLIST:						
	(1) Error analysis performed and budgeted to module.	1II					
	(2) A definitive statement of requirement for accuracy of inputs, outputs, processing, and constants.	1II					
	(3) Sufficiency of math library.			2aII			
	(4) Sufficiency of numerical methods.			2bII		2bII	
	(5) Execution outputs within tolerances.					3 XI	
	METRIC VALUE: Score total from applicable elements / # applicable elements		□		□		□
ERROR TOLERANCE/ CONTROL (p)	ET. 1 ERROR TOLERANCE CONTROL CHECKLIST:						
	(1) Any concurrent processing centrally controlled.			2aII		2aII	
	(2) Errors should be fixable and processing continued. (Errors fixable/total error conditions)				2aII	2aII	
	(3) When an error condition is detected, it should be passed up to calling routine.			2bII		3VII	
	METRIC VALUE: Total score from applicable elements / # applicable elements				□		□

C&IS FORM 140 (13 JAN 77)

FACTOR(S): RELIABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE
INPUT DATA (p)	ET. 2 RECOVERY FROM IMPROPER INPUT DATA CHECKLIST:						
	(1) A definitive statement of requirement for error tolerance of input data.	1 II					
	(2) Range of values (reasonableness) for items specified and checked .			2bII		3IV	
	(3) Conflicting requests and illegal combinations identified and checked.			2b II		3IV	
	(4) All input is checked before processing begins			2b II		3IV	
	(5) Determination that all data is available prior to processing.			2b II		3IV	
	METRIC VALUE: Total score from applicable elements / # applicable elements		□		□		□
RECOVERABLE COMPUTATIONAL FAILURES (p)	ET. 3 RECOVERY FROM COMPUTATIONAL FAILURES CHECKLIST:						
	(1) A definitive statement of requirement for recovery from computational failures.	1 II					
	(2) Loop and multiple transfer index parameters range tested before use.			2bII		3VII	
	(3) Subscript checking.			2bII		3VII	
	(4) Critical output parameters reasonableness checked during processing.			2bII		3VII	
	METRIC VALUE: Total score from applicable elements / # applicable elements		□		□		□

FACTOR(S): RELIABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE
RECOVERABLE HARDWARE FAULTS (a)	ET. 4 RECOVERY FROM HARDWARE FAULTS CHECKLIST: (1) A definitive statement of requirement for recovery from hardware faults. (2) Recovery from hardware faults (e.g., arithmetic faults, power failure, clock).	1 II		2aII		2aII	
	METRIC VALUE: $\frac{\text{Total score from applicable elements}}{\text{\# applicable elements}}$		□		□		□
DEVICE STATUS CONDITIONS (a)	ET. 5 RECOVERY FROM DEVICE ERRORS CHECKLIST: (1) Definitive statement of requirement for recovery from device errors. (2) Recovery from device errors.	1 II		2aII		2aII	
	METRIC VALUE: $\frac{\text{Total score from applicable elements}}{\text{\# applicable elements}}$		□		□		□

FACTOR(S): RELIABILITY, MAINTAINABILITY, TESTABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION		
		YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	
SIMPLICITY/ DESIGN STRUCTURE (p)	SI. 1 DESIGN STRUCTURE MEASURE:							
	(1) Design organized in top down fashion.			2aII		2aII		
	(2) Independence of module	2bIII						
	(3) Module processing not dependent on prior processing		2bII				□	
	(4) Each module description includes input, output processing, limitations.		2bII				□	
	(5) Each module has single entrance, single exit. $\left(\frac{1}{\# \text{ entrances}} + \frac{1}{\# \text{ exists}}\right)$						2bIII	
	(6) Size of data base				2aIX		2aIX	
	(7) Compartamentalization of Data Base $\left(\frac{\text{size}}{\# \text{ files}}\right)$				2aIX		2aIX	
	(8) No duplicate functions.			□		□		d1
	(9) No global data			□		□		d2
	METRIC VALUE: $\frac{\text{Total score from applicable elements}}{\# \text{ applicable elements}}$				□		□	
STRUCTURE PROGRAMMING (a)	SI. 2 USE OF STRUCTURE LANGUAGE OR PREPROCESSOR					3 I		d2
	METRIC VALUE: If used = 1, if not used = 0.						□	

C&IS FORM 140 (13 JAN 77)

FACTOR(S): CONTINUED

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
DATA AND CONTROL FLOW COMPLEXITY (p)	SI. 3 COMPLEXITY MEASURE				2bII		□
	METRIC VALUE: $\frac{\text{Sum of complexity measures for each module}}{\text{\# modules}}$				□		□
CODE SIMPLICITY (p)	SI. 4 MEASURE OF CODING SIMPLICITY						
	(1) Module flow top to bottom.					3 I	
	(2) Negative Boolean or complicated compound Boolean expressions used. $\left(1 - \frac{\text{\# of above}}{\text{\# executable statements}}\right)$						3 I
	(3) Jumps in and out of loops $\frac{(\text{\# single entry/single exit loops})}{\text{total \# loops}}$						3 I
	(4) Loop index modified $\left(1 - \frac{\text{\# loop indices modified}}{\text{total \# loops}}\right)$						3 I
	(5) Module is not self-modifying. $\left(\frac{\text{\# Constracts}}{\text{Total \# LOC}}\right)$						3 I
	(6) Number of statement labels. $\left(1- \frac{\text{\# labels}}{\text{\# executable statements}}\right)$						3 I
	(7) Nesting level $\left(\frac{1}{\text{max nesting level}}\right)$						3 I
	(8) Number of branches $\left(1- \frac{\text{\# branches}}{\text{\# executable statements}}\right)$						3 I
	(9) Number of GOTOs $1- \frac{\text{\# GOTO statements}}{\text{\# executable statements}}$						3 I
	(10) Variable mix in a module $\frac{\text{\# internal variables}}{\text{total \# variables}}$						3VI

FACTOR(S): CONTINUED

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION		
		YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	
	SI. 4 (CONTINUED)							
	(11) Variable density $1 - \frac{\#\ \text{variables}}{\#\ \text{exec statements}}$						3V I	
	(12) All arguments passed to a module are parametric.					□		d3
	(13) Unique names for variables.					□		d1
	(14) Single use of variables.					□		d2
	(15) No mixed mode expressions.					□		d2
	(16) No extraneous code exists.					□		d1
	METRIC VALUE: $\frac{\text{Total score from applicable elements}}{\#\ \text{applicable elements}}$						□	

FACTOR(S): MAINTAINABILITY, FLEXIBILITY, TESTABILITY, PORTABILITY, REUSABILITY, INTEROPERABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION		
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	
MODULARITY/ DEGREE OF INDEPENDENCE	NO. 1 STABILITY MEASURE $\left(\frac{\text{Expected \# modules changed}}{\text{total \# modules}}\right)$							d1
	METRIC VALUE: Same as entry above				□			
MODULAR IMPLEMENTATION (p)	NO. 2 MODULAR IMPLEMENTATION MEASURE							
	(1) Hierarchical structure $\left(1-\frac{\text{\# violations of hierarchy}}{\text{total \# modules}}\right)$				2aII		2aII	
	(2) Module Size Profile						3 I	m
	(3) Controlling parameters defined by calling module $\frac{\text{\# control variables}}{\text{\# calling parameters}}$				2bIV		3 V	m
	(4) Input data controlled by calling module			2bIV			3 V	m
	(5) Output data provided to calling module			2bIV			3 V	m
	(6) Control returned to calling module			2bIV			□	m
	(7) Modules do not share temporary storage			2bIV		3bIV		
	(8) All modules represent one function $\left(1-\frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$				□		□	d1
	METRIC VALUE = $\left(\frac{\text{Total score from applicable elements}}{\text{\# applicable elements}}\right)$				□		□	

FACTOR(S): FLEXIBILITY, REUSABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION		
		YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	
GENERALITY/ REFERENCES (p)	GE. 1 EXTENT TO WHICH MODULE IS REFERENCED BY OTHER MODULES $\left(\frac{\text{\# common modules}}{\text{total \# modules}}\right)$				2aIII		2aII	
	METRIC VALUE: Same as line above				□		□	
IMPLEMENTATION GENERALITY (p)	GE. 2 IMPLEMENTATION FOR GENERALITY CHECKLIST							
	(1) Input, processing, output functions are not mixed in a single module.			2bIV			3IV	m
	(2) Application and machine-dependent functions are not mixed in a single module. $\left(\frac{1}{\text{\# Machine dependent functions}}\right)$				2bIV		3IX	m
	(3) Processing not data volume limited			2bIV			3 X	m
	(4) Processing not data value limited			2bIV			3 X	m
	(5) All constants should be defined once $\left(1-\frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$						□	d2
	METRIC VALUE = $\left(\frac{\text{Total score from applicable elements}}{\text{\# applicable elements}}\right)$				□		□	

FACTOR(S): FLEXIBILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
EXPANDABILITY/ DATA STORAGE EXPANSION (a)	EX.1 DATA STORAGE EXPANSION MEASURE:						
	(1) Logical processing independent of storage specification/requirements (by module) $1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}$				2bV		2bV
	(2) Percent of memory capacity uncommitted $\frac{\text{Amount of memory uncommitted}}{\text{Total amount of available memory}}$						3 X
	METRIC VALUE = $\frac{\text{Total score from applicable elements}}{\text{\# applicable elements}}$				□		□
COMPUTATION EXTENSIBILITY (a)	EX.2 EXTENSIBILITY MEASURE:						
	(1) Accuracy, convergence, timing attributes which control processing are parametric $1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}$				2bV		3 X
	(2) Modules table driven $1 - \frac{\text{\# modules not table driven}}{\text{total \# modules}}$				2bV		3 X
	(3) Percent of speed capacity uncommitted $\frac{\text{Amount of cycle time uncommitted}}{\text{total processing time}}$						3XI
	METRIC VALUE = $\frac{\text{Total score from applicable elements}}{\text{\# applicable elements}}$				□		□

FACTOR(S): TESTABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE
INSTRUMENTATION/ MODULE TESTING SUPPORT (a)	IN. 1 MODULE TESTING MEASURE (by module) (1) Path coverage $\frac{\text{\# paths to be tested}}{\text{total \# paths}}$ (2) All input parameters boundary tested $\frac{\text{\# parameters to be boundary tested}}{\text{total \# parameters}}$				2bVIII 2aVIII		2aVII 2aVII
	METRIC VALUE = $\frac{\text{Sum of module testing measures for each module}}{\text{total \# modules}}$				□		□
INTEGRATION TESTING SUPPORT (a)	IN. 2 INTEGRATION TESTING MEASURE (1) Module interfaces tested $\frac{\text{\# to be tested}}{\text{total \# interfaces}}$ (2) Performance requirements (timing & storage) coverage $\frac{\text{\# requirements to be tested}}{\text{total \# perf requirements}}$				2aVIII 2aVIII		2aVII 2aVII
	METRIC VALUE = $\frac{\text{Total score from applicable elements}}{\text{\# applicable elements}}$				□		□
SYSTEM TESTING SUPPORT (a)	IN. 3 SYSTEM TESTING MEASURE (1) Module coverage (for all test scenarios) $\frac{\text{\# modules to be executed}}{\text{total \# of modules}}$ (2) Identification of test inputs and outputs in summary form			2aVIII	2aVIII	2aVII	2aVII
	METRIC VALUE = $\frac{\text{Total score from applicable elements}}{\text{\# applicable elements}}$				□		□

FACTOR(S): FLEXIBILITY, MAINTAINABILITY, TESTABILITY PORTABILITY, REUSABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
SELF-DESCRIPTIVENESS/QUANTITY OF COMMENTS (p)	SD. 1 QUANTITY OF COMMENTS (by module) $\frac{\text{\# of comments (nonblank)}}{\text{Total \# lines (nonblank)}}$						3III
	METRIC VALUE = $\frac{\text{Sum of quantity of comment measures for each module}}{\text{total \# modules}}$						□
EFFECTIVENESS OF COMMENTS (p)	SD. 2 EFFECTIVENESS OF COMMENTS MEASURE (1) Modules have standard formated prologue comments which describe: - Module name/version number - Author - Date - Purpose - Inputs - Outputs - Function - Assumptions - Limitations and restrictions - Accuracy requirements - Error recovery procedures - References $1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}$						3III
	(2) Comments set off from code in uniform manner $1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}$						3III
	(3) All transfers of control & destinations commented $1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}$						3III
	(4) All machine dependent code commented $1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}$						3III

FACTOR(S): FLEXIBILITY, MAINTAINABILITY, TESTABILITY, PORTABILITY, REUSABILITY (CONTINUED)

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
	(5) All non-standard HOL statements commented $1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ module}}$						□
	(6) Attributes of all declared variables commented $1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ modules}}$						□
	(7) Comments do not just repeat operation described in language $1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ modules}}$						□
	SYSTEM METRIC VALUE = $\frac{\text{Total scores from applicable elements}}{\#\text{ applicable elements}}$						□
DESCRIPTIVENESS OF IMPLEMENTATION LANGUAGE	SD. 3 DESCRIPTIVENESS OF IMPLEMENTATION LANGUAGE MEASURE						
	(1) High order language used $1-\frac{\#\text{ modules with direct code}}{\text{total }\#\text{ modules}}$						□
	(2) Variable names (mnemonic) descriptive of physical or functional property represented $1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ modules}}$						□
	(3) Source code logically blocked and indented $1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ modules}}$						□
	(4) One statement per line $1-\frac{\#\text{ continuations + multiple statement lines}}{\text{total }\#\text{ lines}}$						□ m
	(5) Standard format for organization of modules followed $1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ modules}}$						□ d3

FACTOR(S): FLEXIBILITY, MAINTAINABILITY,
TESTABILITY, PORTABILITY,
REUSABILITY (CONTINUED)

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
	(6) No language keywords used as names $1- \frac{\text{\# modules violate rule}}{\text{total \# modules}}$						□ d1
	SYSTEM METRIC VALUE = $\frac{\text{Total score from applicable elements}}{\text{\# applicable elements}}$						□

FACTOR(S): EFFICIENCY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
EXECUTION EFFICIENCY/ REQUIREMENTS (a)	EE. 1 PERFORMANCE REQUIREMENTS IDENTIFIED AND ALLOCATED TO DESIGN	IV		2bVI			
	METRIC VALUE = Same as line above				□		
ITERATIVE PROCESSING (a)	EE. 2 ITERATIVE PROCESSING EFFICIENCY MEASURE: (by module)						
	(1) Non-loop dependent computations kept out of loop $1 - \frac{\text{\# nonloop dependent statements in loop}}{\text{total \# loop statements}}$				2bVI		3VII
	(2) Performance optimizing compiler/assembly language used					2bVI	
	(3) Compound expressions defined once $1 - \frac{\text{\# compound expression defined more than once}}{\text{\# compound expressions}}$						3VII
	(4) Number of overlays $\frac{1}{\text{\# of overlays}}$				2aIV		2aIV
	(5) Free of bit/byte packing/unpacking in loops			2bVI		3VI	
	(6) Module linkages $1 - \frac{\text{module linkage time}}{\text{execution time}}$						3XI
	(7) OS linkages $1 - \frac{\text{OS linkage time}}{\text{execution time}}$						3XI

FACTOR(S): EFFICIENCY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION		
		YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	
	(8) Efficient use of storage facility			□				
	(9) Free of nonfunctional executable code						□	d1
	$1-\frac{\#\text{ nonfunctional executable code}}{\text{total executable statements}}$							
	(10) Decision statements efficiently coded						□	d1
	$1-\frac{\#\text{ inefficient decision statements}}{\text{Total }\#\text{ decision statements}}$							
	METRIC VALUE = $\frac{\text{total score from applicable elements}}{\text{total }\#\text{ applicable elements}}$				□		□	
DATA USAGE (a)	EE. 3 DATA USAGE EFFICIENCY MEASURE: (by module)							
	(1) Data grouped for efficient processing			2aIV		2aIV		
	(2) Variables initialized when declared						3VII	
	$\frac{\#\text{ initialized when declared}}{\text{total }\#\text{ variables}}$							
	(3) No mix-mode expressions						3VII	
	$1-\frac{\#\text{ mix mode expressions}}{\#\text{ executable statements}}$							
	(4) Common choice of units/type						3VI	m
	1/# occurrences of uncommon unit operations							
	(5) Data indexed or referenced for efficient processing			bVI		2bVI		
	(6) Static data $\frac{\#\text{ static data items}}{\text{data base size}}$						3XI	
	(7) Dynamic data $\frac{\#\text{ modified data items}}{\text{data base size}}$						3XI	
	MODULE METRIC VALUE = $\frac{\text{total score from applicable elements}}{\#\text{ applicable elements}}$				□		□	

FACTOR(S): EFFICIENCY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
STORAGE EFFICIENCY (a)	SE. 1 STORAGE EFFICIENCY MEASURE: (by module)						
	(1) Storage requirements allocated to design			2aIV			
	(2) Virtual storage facilities used			2aIV		2aIV	
	(3) Common data defined only once						2aIV
	$1-\frac{\#\text{ variables defined more than once}}{\text{total }\#\text{ variables}}$						
	(4) Program segmentation				2aIV		2aIV
	$1-\frac{\text{maximum segment length}}{\text{total program length}}$						
	(5) Dynamic memory management utilized			2aIV		2aIV	
	(6) Data packing used					3VIII	
	(7) Storage optimizing compiler/assembly language used			2aIV		2aIV	
	(8) Free of nonfunctional code						□ d1
	$1-\frac{\#\text{ nonfunctional statements}}{\text{total }\#\text{ statements}}$						
	(9) No duplicate codes				□		□ d1
	$1-\frac{\#\text{ duplicate statements}}{\text{total }\#\text{ statements}}$						
	(10) Data segmentation				□		□ d1
	$1-\frac{\text{Amount of unused data}}{\text{total amount of data}}$						
	(11) Free of redundant data elements						□ d1
	$1-\frac{\#\text{ redundant data elements}}{\#\text{ data elements}}$						
	METRIC VALUE = $\frac{\text{total score from applicable elements}}{\#\text{ applicable elements}}$				□		□
					□		□

FACTOR(S): INTEGRITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
ACCESS CONTROL (a)	AC.1 ACCESS CONTROL CHECKLIST:						
	(1) User I/O access controls provided (ID's, passwords)	1111		2aV		2aV	
	(2) Data base access controls provided (authorization tables, privacy locks)	1111		2aV		2aV	
	(3) Memory protection across tasks provided	1111		2aV		2aV	
	METRIC VALUE = $\frac{\text{total score from applicable elements}}{\text{\# applicable elements}}$		□		□		□
ACCESS AUDIT (a)	AA.1 ACCESS AUDIT CHECKLIST:						
	(1) Provisions for recording and reporting access	1111		2aV		2aV	
	(2) Provisions for immediate indication of access violation	1111		2aV		2aV	
	METRIC VALUE = $\frac{\text{total score from applicable elements}}{\text{\# applicable elements}}$		□		□		□

C&IS FORM 140 (13 JAN 77)

FACTOR(S): USABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE
OPERABILITY	OP 1 OPERABILITY CHECKLIST:						
(a)	(1) All steps of operation described (normal and alternative flows)	1IV		2aVII		2aVI	
	(2) All error conditions and responses appropriately described to operator	1IV		2aVII		2aVI	
	(3) Provisions for operator to interrupt, obtain status, save, modify, and continue processing	1IV		2aVII		2aVI	
	(4) Number of operator actions reasonable $1 - \frac{\text{time for operator actions}}{\text{total time for job}}$						2aVII
	(5) Job set up and tear down procedures described					2aVI	
	(6) Hard copy log of interactions maintained			2aVII		2aVI	
	(7) Operator messages consistent and responses standard			2aVII		2aVI	
	METRIC VALUE = $\frac{\text{total score from applicable elements}}{\text{\# applicable elements}}$		□		□		□
TRAINING	TN. 1 TRAINING CHECKLIST:						
(a)	(1) Lesson plans/training material developed for operators, end users, maintainers					2aVI	
	(2) Realistic simulated exercises provided					2aVI	
	(3) Sufficient 'help' and diagnostic information available on-line			2aVII		2aVI	
	METRIC VALUE = $\frac{\text{total score from applicable elements}}{\text{\# applicable elements}}$				□		□

FACTOR(S): USABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE
COMMUNICATIVENESS/ USER INPUT INTERFACE	CM. 1 USER INPUT INTERFACE MEASURE:						
(a)	(1) Default values defined $\frac{\text{\# defaults}}{\text{total \# parameters}}$				2aVII		2aVII
	(2) Input formats uniform $\frac{1}{\text{\# different input record formats}}$				2aVII		2aVII
	(3) Each input record self identifying $1 - \frac{\text{\# that are not self identifying}}{\text{total \# input records}}$				2aVII		2aVII
	(4) Input can be verified by user prior to execution			2aVII		2aVIII	
	(5) Input terminated by explicitly defined logical end of input			2aVII		2aVIII	
	(6) Provision for specifying input from different media	IV		2aVII		2aVIII	
	METRIC VALUE: $\frac{\text{total score from applicable elements}}{\text{\# applicable elements}}$		□		□		□
USER OUTPUT INTERFACE	CM. 2 USER OUTPUT INTERFACE MEASURE:						
(a)	(1) Selective output controls	1IV		2aVIII		2aVI	
	(2) Outputs have unique descriptive user oriented lables			2aVIII		2aVII	
	(3) Outputs have user oriented units			2aVIII		2aVII	
	(4) Uniform output formats $\frac{1}{\text{\# different output formats}}$				2aVII		2aVII
	(5) Logical groups of output separated for user examination			2aVII		2aVII	

FACTOR(S): USABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR ø	VALUE	YES/NO 1 OR ø	VALUE	YES/NO 1 OR ø	VALUE
	(6) Relationship between error messages and outputs is unambiguous			2aVII		2aVII	
	(7) Provision for reducing output to different media	1IV		2aVII		2aVII	
	METRIC VALUE = $\frac{\text{total score from applicable elements}}{\text{\# applicable elements}}$		□		□		□

FACTOR(S): PORTABILITY, REUSABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE
SOFTWARE SYSTEM INDEPENDENCE/ (p)	SS. 1 SOFTWARE SYSTEM INDEPENDENCE MEASURE:						
	(1) Dependence on software system utility programs, system library routines, and other system facilities $\left(\frac{\text{\# system references}}{\text{Total LOC}}\right)$				2bIV		3 V
	(2) Common, standard subset of language used $1-\frac{\text{\# module violate rule}}{\text{total \# modules}}$				2bIV		3 III
	(3) Dependence on software system library routines $1-\frac{\text{\# library routines used}}{\text{total \# modules}}$				□		□ d3
	(4) Free from operating system references $1-\frac{\text{\# modules with OS references}}{\text{total \# modules}}$				□		□ d3
	METRIC VALUE = $\frac{\text{total score from applicable elements}}{\text{\# applicable elements}}$				□		□

C&IS FORM 140 (13 JAN 77)

FACTOR(S): PORTABILITY, REUSABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE
MACHINE INDEPENDENCE (p)	MI. 1 MACHINE INDEPENDENCE MEASURE:						
	(1) Programming language used available on other machines			2bIV		2bIV	
	(2) Free from input/output references $\left(\frac{\text{\# I/O references}}{\text{Total LOC}}\right)$				2bIV		3IV m
	(3) Code is independent of word and character siz $1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}$						3IX
	(4) Data representation machine independent $1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}$						3IX
	MODULE VALUE= $\frac{\text{total score from applicable elements}}{\text{\# applicable elements}}$				□		□

FACTOR(S): INTEROPERABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS YES/NO 1 OR Ø	REQMTS VALUE	DESIGN YES/NO 1 OR Ø	DESIGN VALUE	IMPLEMENTATION YES/NO 1 OR Ø	IMPLEMENTATION VALUE
COMMUNICATIONS COMMONALITY (a)	CC. 1 COMMUNICATIONS COMMONALITY CHECKLIST:						
	(1) Definitive statement of requirement for communication with other systems	IVII					
	(2) Protocol standards established and followed			2aVI		2aVI	
	(3) Single module interface for input $\frac{1}{\#\text{ modules used for input}}$				2aVI		2aVI
	(4) Single module interface for output $\frac{1}{\#\text{ modules used for output}}$				2aVI		2aVI
	METRIC VALUE = $\frac{\text{total score from applicable elements}}{\#\text{ applicable elements}}$		□		□		□
DATA COMMONALITY	DC. 1 DATA COMMONALITY CHECKLIST:						
	(1) Definitive statement for standard data representation for communication with other systems	1VI					
	(2) Translation standards among ıepresentations established and followed			2aVI		2aVI	
	(3) Single module to perform each translation $\frac{1}{\#\text{ modules used to perform translation}}$				2aVI		2aVI
	METRIC VALUE = $\frac{\text{total score from applicable elements}}{\#\text{ applicable elements}}$		□		□		□

FACTOR(S): MAINTAINABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE
CONCISENESS (p)	CO. 1 HALSTEAD'S MEASURE (by module) $1 - \left\lvert \frac{\text{module length calculated-module length observed}}{\text{module length calculated}} \right\rvert$						3II
	METRIC VALUE $= \left\lvert \frac{\text{module length calculated-module length observed}}{\text{module length calculated}} \right\rvert$						□

EXPLANATIONS OF METRICS

Each metric and each metric element are described in the following paragraphs. Indication is provided if the metric is applied at the system level or the module level and during which phases.

Traceability

TR.1 Cross reference relating modules to requirements (design and implementation phases at system level).

During design, the identification of which itemized requirements are satisfied in the design of a module are documented. A traceability matrix is an example of how this can be done. During implementation, which itemized requirements are being satisfied by the module implementation are to be identified. Some form of automated notation, prologue comments or imbedded comments, is used to provide this cross reference. The metric is the identification of a tracing from requirements to design to code,

Completeness

CP.1 Completeness Checklist (All three phases at system level).

This metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) Unambiguous references (input, function, output).
Unique references to data or functions avoid ambiguities such as a function being called one name by one module and by another name by another module. Unique references avoid this type of ambiguity in all three phases.

(2) All data references defined, computed, or obtained from an external source.
Each data element is to have a specific origin. At the requirements level only major global data elements and a few specific local data elements may be available to be checked. The set of data elements available for completeness checking at the design level increases substantially and is to be complete at implementation.

(3) All defined functions used.
A function which is defined but not used during a phase is either nonfunctional or a reference to it has been omitted.

(4) All referenced functions defined.
A system is not complete at any phase if dummy functions are present or if functions have been referenced but not defined.

(5) All conditions and processing defined for each decision point.
Each decision point is to have all of its conditions and alternative processing paths defined at each phase of the software development. The level of detail to which the conditions and alternative processing are described may vary but the important element is that all alternatives are described.

(6) All defined and referenced calling sequence parameters agree.
For each interaction between modules, the full complement of defined parameters for the interface is to be used. A particular call to a module should not pass, for example, only five of the six defined parameters for that module.

(7) All problem reports resolved.
At each phase in the development, problem reports are generated. Each is to be closed or a resolution indicated to ensure a complete product.

Consistency

CS.1 Procedure Consistency Measure (design and implementation at system level).

The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) Standard Design Representation.
Flow charts, HIPO charts, Program Design Language - whichever form of design representation is used, standards for representing the elements of control flow are to be established and followed. This element applies to design only. The measure is based on the number of modules whose design representation does not comply with the standards.

(2) Calling sequence conventions.
Interactions between modules are to be standardized. The standards are to be established during design and followed during implementation. The measure is based on the number of modules which do not comply with the conventions.

(3) Input/Output Conventions.
Conventions for which modules will perform I/O, how it will be accomplished, and the I/O foımats are to be established and followed. The measure is based on which modules do not comply with the conventions.

(4) Error Handling Conventions.
A consistent method for error handling is required. Conventions established in design are followed into implementation. The measure is based on the number of modules which do not comply with the conventions.

CS.2 Data Consistency Measure (Design and implementation at system level)
The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) Standard data usage representation.
In concert with CS.1 (1), a standard design representation for data usage is to be established and followed. This is a design metric only, identifying the number of modules which violate the standards.

(2) Naming Conventions.
Naming conventions for variables and modules are to be established and followed.

(3) Consistent Global Definitions.
Global data elements are to be defined in the same manner by all modules. The measure is based on the number of modules in which the global data elements are defined in an inconsistent manner for both design and implementation.

Accuracy

AC.1 Accuracy Checklist (requirements, design, implementation phases at system level). Each element is a binary measure indicating existence, or absence of the elements. The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) Error analysis performed and budgeted to module (requirements phase only).
An error analysis must be part of the requirements analysis performed to develop the requirements specification. This analysis allocates overall accuracy requirements to the individual functions to be performed by the system. This budgeting of accuracy requirements provides definitive objectives to the module designers and implementers.

(2) A definitive statement of requirement for accuracy of inputs, outputs, processing, and constants (requirements phase only).
See explanation above (1).

(3) Sufficiency of Math Library (design phase only).
The accuracy of the math library routines utilized within the system is to be checked for consistency with the overall accuracy objectives.

(4) Sufficiency of numerical methods (design and implementation phase).
The numerical methods utilized within the system are to be consistent with the accuracy objectives. They can be checked at design and implementation.

(5) Execution outputs within tolerances (implementation phase only requiring execution).
A final measure during development testing is execution of modules and checking for accuracy of outputs.

Error Tolerance

ET.1 Error Tolerance Control Checklist (design and implementation phases at system level).

The metric is the sum of the scores given to the following elements divided by the number of applicable elements.

(1) Concurrent processing centrally controlled.
Functions which may be used concurrently are to be controlled centrally to provide concurrency checking, read/write locks, etc. Examples are a data base manager, I/O handling, error handling, etc. The central control must be considered at design and then implemented.

(2) Errors fixable and processing continued.
When an error is detected, the capability to correct it on-line and then continue processing, should be available. An example is an operator message that the wrong tape is mounted and processing will continue when correct tape is mounted. This can be measured at design and implementation.

(3) When an error condition is detected, the condition is to be passed up to calling routine.
The decision of what to do about an error is to be made at a level where an affected module is controlled. This concept is built into the design and then implemented.

ET.2 Recovery from Improper Input Data Checklist (all three phases at system level). The metric is the sum of the scores of the following applicable elements divided by the number of the applicable elements.

(1) A definitive statement of requirement for error tolerance of input data.
The requirements specification must identify the error tolerance capabilities desired (requirements phase only).

(2) Range of values (reasonableness) for items specified and checked (design and implementation phases only).
The attributes of each input item are to be checked for reasonableness. Examples are checking items if they must be numeric, alphabetic, positive or negative, of a certain length, nonzero, etc. These checks are to be specified at design and exist in code at implementation.

(3) Conflicting requests and illegal combinations identified and checked (design and implementation phases only).
Checks to see if redundant input data agrees, if combinations of parameters are reasonable, and if requests are conflicting should be documented in the design and exist in the code at implementation.

(4) All input is checked before processing begins (design and implementation phases only).
Input checking is not to stop at the first error encountered but to continue through all the input and then report all errors. Processing is not to start until the errors are reported and either corrections are made or a continue processing command is given.

(5) Determination that all data is available prior to processing.
To avoid going through several processing steps before incomplete input data is discovered, checks for sufficiency of input data are to be made prior to the start of processing.

ET.3 Recovery from Computational Failures Checklist (all three phases at system level). The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) A definitive statement of requirement for recovery from computational failures (requirements phase only).
The requirement for this type error tolerance capability are to be stated during requirements phase.

(2) Loop and multiple transfer index parameters range tested before use. (design and implementation phase only).
Range tests for loop indices and multiple transfers are to be specified at design and to exist in code at implementation.

(3) Subscript checking (design and implementation phases only).
Checks for legal subscript values are to be specified at design and coded during implementation.

(4) Critical output parameters reasonableness checked during processing (design and implementation phases only).
Certain range-of-value checks are to be made during processing to ensure the reasonableness of final outputs. This is usually done only for critical parameters. These are to be identified during design and coded during implementation.

ET.4 Recovery from Hardware Faults Checklist (All three phases at system level). The metric is the sum of scores from the applicable elements divided by the number of applicable elements.

(1) A definitive statement of requirements for recovery from hardware faults (requirements only).
The handling of hardware faults such as arithmetic faults, power failure, clock interrupts, etc., are to be specified during requirements phase.

(2) Recovery from Hardware Faults (design and implementation phases only).

The design specification and code to provide the recovery from the hardware faults identified in the requirements must exist in the design and implementation phases respectively.

ET.5 Recovery from Device Errors Checklist (all three phases at system level). The metric is the score given to the applicable elements below at each phase.

(1) A definitive statement of requirements for recovery from device errors (requirements only).

The handling of device errors such as unexpected end-of-files or end-of-tape conditions or read/write failures are specified during the requirements phase.

(2) Recovery from Device Errors (design and implementation phases only).

The design specification and code to provide the required handling of device errors must exist in the design and implementation phases respectively.

Simplicity

SI.1 Design Structure Measure (design and implementation phases at system level). The metric is the sum of the scores of the applicable elements divided by the number of applicable elements.

(1) Design organized in top down fashion.

A hierarchy chart of system modules is usually available or easy to construct from design documentation. It should reflect the accepted notion of top down design. The system is organized in a hieracrchal tree structure, each level of the tree represents lower levels of detail descriptions of the processing.

(2) Module independence.
The processing done within a module is not to be dependent on the source of input or the destination of the output. This rule can be applied to the module description during design and the coded module during implementation. The measure for this element is based on the number of modules which do not comply with this rule.

(3) Module processing not dependent on prior processing.
The processing done within a module is not to be dependent upon knowledge or results of prior processing, e.g., the first time through the module, the nth time through, etc. This rule is applied as above at design and implementation.

(4) Each module description includes input, output, processing, limitations.
Documentation which describes the input, output, processing, and limitations for each module is to be developed during design and available during implementation. The measure for this element is based on the number of modules which do not have this information documented.

(5) Each module has single entrance, single exit.
Determination of the number of modules that violate this rule at design and implementation can be made and is the basis for the metric.

(6) Size of data base.
The size of the data base in terms of the number of unique data items contained in the data base relates to the design structure of the software system. A data item is a unique data element for example an individual data entry or data field.

(7) Compartamentalization of Data Base

The structure of the data base also is represented by its modularization or how it is decomposed. The size determined in (6) above divided by the number of data sets provided this measure. A data set ccrresponds to the first level of decomposition of a data oase, e.g., a set in a CODASYL data base, a record in a file system, a COMMON in FORTRAN, or a Data Block in a COMPOOL system

SI.3 Data and Control Flow Complexity measure (Design and implementation phases).

This metric can be measured from the design representation (e.g., flowcharts) and the code automatically. Path flow analysis and variable set/use information along each path is utilized. A variable is considered to be 'live' at a node if it can be used again along that path in the program. The complexity measure is based on summing the 'liveness' of all variables along all paths in the program. It is normalized by dividing it by the maximum complexity of the program (all variables live along all paths). (See [RICHP76] and page D-16 of Volume II.)

SI.4 Measure of Simplicity of Coding Techniques (Implementation phase applied at module level first). The metric at the system level is an averaged quantity of all the module measures for the system. The module measure is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) Module flow top to bottom.

This is a binary measure of the logic flow of a module. If it flows top to bottom, it is given a value of 1, if not a 0.

(2) Negative Boolean or complicated Compound Boolean expressions used.

Compound expressions involving two or more Boolean operators and negation can often be avoided. These types of expressions add to the complexity of the module. The measure is based on the number of these complicated expressions per executable statement in the module.

(3) Jumps in and out of loops.

Loops within a module should have one entrance and one exit. This measure is based on the number of loops which comply with this rule divided by the total number of loops.

(4) Loop index modified.

Modification of a loop index not only complicates the logic of a module but causes severe problems while debugging. This measure is based on the number of loop indices which are modified divided by the total number of loops.

(5) Module is not self-modifying.

If a module has the capability to modify its processing logic it becomes very difficult to recognize what state it is in when an error occurs. In addition, static analysis of the logic is more difficult. This measure emphasizes the added complexity of self-modifying modules.

(6) Number of statement labels.

This measure is based on the premise that as more statement labels are added to a module the more complex it becomes to understand.

(7) Nesting level.

The greater the nesting level of decisions or loops within a module, the greater the complexity. The measure is the inverse of the maximum nesting level.

(8) Number of branches.

The more paths or branches that are present in a module, the greater the complexity. This measure is based on the number of decision statements per executable statements.

(9) Number of GOTO's.

Much has been written in the literature about the virtues of avoiding GOTO's. This measure is based on the number of GOTO statements per executable statement.

(10) Variable mix in a module.

From a somplicity viewpoint, local variables are far better than global variables. This measure is the ratio of internal (local) variables to total (internal (local) plus external (global)) varialbes within a module.

(11) Variable density.

The more used of variables in a module the greater the complexity of that module. This measure is based on the number of variable uses in a module divided by the maximum possible uses.

Modularity

MO.2 Modular Implementation Measure (design and implementation phases at system level). The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) Hierarchical Structure.

The measure refers to the modular implementation of the top down design structure mentioned in SI.1 (1). The hierarchical structure obtained should exemplify the following rules: Interactions between modules are restricted to flow of control between a predecessor module and its immediate successor modules. This measure is based on the number of violations to this rule.

(2) Module Size Profile.

The standard module size of procedural statements can vary. 100 statements has been mentioned in the literature frequently. This measure is based on the number of procedural statements in a module.

(3) Controlling parameters defined by calling module.

The next four elements further elaborate on the control and interaction between modules referred to by (1) above. The calling module defines the controlling parameters, any input data required, and the output data required. Control must also be returned to the calling module. This measure is based on the number of calling parameters which are control parameters. The next three are based on whether a rule is violated. They can be measured at design and implementation.

(4) Input data controlled by calling module.

See (3) above.

(5) Output data provided to calling module.

See (3) above.

(6) Control returned to calling module.

See (3) above.

(7) Modules do not share temporary storage.

This is a binary measure, 1 if modules do not share temporary storage and 0 if they do. It emphasizes the loss of module independence if temporary storage is shared between modules.

Generality

GE.1 Extent to which modules are referenced by other modules (design and implementation at system level). This metric provides a measure of the generality of the modules as they are used in the current system. A module is considered to be more general in nature if it is used (referenced) by more than one module. The number of these common modules divided by the total number of modules provides the measure.

GE. 2 Implementation for Generality Measure (design and implementation phases). This metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) Input, processing, output functions are not mixed in a single function.
A module which performs I/O as well as processing is not as general as a module which simply accomplishes the processing. This measure is based on the number of modules that violate this concept at design and implementation.

(2) Application and machine dependent functions are not mixed in a single module (implementation only).
Any references to machine dependent functions within a module lessens its generality. An example would be referencing the system clock for timing purposes. This measure is based on the number of machine dependent functions in a module.

(3) Processing not data volume limited.
A module which has been designed and coded to accept no more than 100 data item inputs for processing is certainly not as general in nature as a module which will accept any volume of input. This measure is based on the number of modules which are designed or implemented to be data volume limited.

(4) Processing not data value limited.
A previously identified element, ET.2 (2) of Error Tolerance dealt with checking input for reasonableness. This capability is required to prevent providing data to a function for which it is not defined or its degree of precision is not acceptable, etc. This is necessary capability from an error tolerance viewpoint. From a generality viewpoint, the smaller the subset

of all possible inputs to which a function can be applied the less general it is. Thus, this measure is based on the number of modules which are data value limited. This can be determined at design and implementation.

Expandability

EX.1 Data Storage Expansi n Measure (design and implementation phase at system level). The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) Logical processing independent of storage specification/requirements. The logical processing of a module is to be independent of storage size, buffer space, or array sizes. The design provides for variable dimensions and dynamic array sizes to be defined parametrically. The metric is based on the number of modules containing hard-coded dimensions which do not exemplify this concept.

(2) Percent of memory capacity uncommitted (implementation only). The amount of memory available for expansion is an important measure. This measure identifies the percent of available memory which has not been utilized in implementing the current system.

EX.2 Extensibility Measure (design and implementation phases at the system level). The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) Accuracy, convergence, timing attributes which control processing are parametric.

A module which can provide varying degrees of convergence or timing to achieve greater precision provides this attribute of extensibility. Hard-coded control parameters, counters, clock values, etc. violate this measure. This measure is based on the number of modules which do not exemplify this characteristic. A determination can be made during design and implementation.

(2) Modules table driven.

The use of tables within a module facilitates different representations and processing characteristics. This measure which can be applied during design and implementation is based on the number of modules which are not table driven.

(3) Percent of speed capacity uncommitted (implementation only).

A certain function may be required in the performance requirements specification to be accomplished in a specified time for overall timing objectives. The amount of time not used by the current implementation of the function is processing time available for potential expansion of computational capabilities. This measure identifies the percent of total processing time that is uncommitted.

Instrumentation

IN.1 Module testing measure (design and implementation phases, first at module level then system level). The system level metric is an average of all module measures. The module measure is the average score of the following two elements:

(1) Path coverage.

Plans for testing the various paths within a module should be made during design and the test cases actually developed during implementation. This measure identifies the number of paths planned to be tested divided by the total number of paths.

(2) Input parameters boundary tested.

The other aspect of module testing involves testing the input ranges to the module. This is done by exercising the module at the various boundary values of the input parameters. Plans to do this must be specified during design and coded during implementation. The measure is the number of parameters to be boundary tested divided by the total number of parameters.

IN.2 Integration Testing Measure (design and implementation phases at system level). The metric is the averaged score of the following two elements.

(1) Module interfaces tested.
One aspect of integration testing is the testing of all module to module interfaces. Plans to accomplish this testing are prepared during design and the tests are developed during implementation. The measure is based on the number of interfaces to be tested divided by the total number of interfaces.

(2) Performance requirements (timing and storage) coverage.
The second aspect of integration testing involves checking for compliance at the module and subsystem level with the performance requirements. This testing is planned during design and the tests are developed during implementation. The measure is the number of performance requirements to be tested divided by the total number of performance requirements.

IN.3 System Testing Measure (design and implementation phases at the system level). The metric is the averaged score of the two elements below.

(1) Module Coverage.
One aspect of system testing which can be measured as early as the design phase is the equivalent to path coverage at the module level. For all system test scenarios planned, the percent of all of the modules to be exercised is important.

(2) Identification of test inputs and outputs in summary form.
The results of tests and the manner in which these results are displayed are very important to the effectiveness of testing. This is especially true during system testing because of the potentially large volume of input and output data. This measure simply identifies if the capability exists to display test inputs and outputs in a summary fashion. The measure can be applied to the plans and specifications in the design phase and the development of this capability during implementation.

Self Descriptiveness

SD.1 Quantity of Comments (implementation phase at module level first and then system level). The metric is the number of comment lines divided by the total number of lines in each module. Blank lines are not counted. The average value is computed for the system level metric.

SD.2 Effectiveness of Comments Measure (implementation phase at system level). The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) Modules have standard formatted prologue comments.
The items to be contained in the prologue comments are listed in Table 6.2-1. This information is extremely valuable to new personnel who have to work with the software after development, performing maintenance, testing, changes, etc. The measure at the system level is based on the number of modules which do not comply with a standard format or do not provide complete information.

(2) Comments set off from code in uniform manner.
Blank lines, bordering asterisks, specific card columns are some of the techniques utilized to aid in the identification of comments. The measure is based on the number of modules which do not follow the conventions established for setting off the comments.

(3) All transfers of control and destinations commented.
This form of comment aids in the understanding and ability to follow the logic of the module. The measure is based on the number of modules which do not comply.

(4) All machine dependent code commented.
Comments associated with machine dependent code are important not only to explain what is being done but also serves to identify that portion of the module as machine dependent. The metric is based on the number of modules which do not have the machine dependent code commented.

(5) All non-standard HOL statements commented.
A similar explanation to (4) above is applicable here.

(6) Att^ributes of all declared variables commented.

The usage, properties, units, etc., of variables are to be explained in comments. The measure is based on the number of modules which do not follow this practice.

(7) Comments do not just repeat operation described in language.

Comments are to describe why not what. A comment, increment A by 1, for the statement A=A+1 provides no new information. A comment, increment the table look-up index, is more valuable for understanding the logic of the module. The measure is based on the number of modules in which comments do not explain the why's.

SD.3 Descriptiveness of Implementation Language Measure (implementation phase at system level). The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) High Order Language used.

An HOL is much more self-descriptive than assembly language. The measure is based on the number of modules which are implemented, in whole or part, in assembly or machine language.

(2) Variable names (mnemonics) descriptive of physical or functional property represented.

While the metric appears very subjective, it is quite easy to identify if variable names have been chosen with self-descriptiveness in mind. Three variable names such as NAME, POSIT, SALRY are far better and more easily recognized as better than A1, A2, A3. The measure is based on the number of modules which do not utilize descriptive names.

(3) Source code logically blocked and indented.
Techniques such as blocking, paragraphing, indenting for specific constructs are well established and are to be followed uniformly with a system. This measure is based on the number of modules which do not comply with a uniform technique.

(4) One statement per line.
The use of continuation statements and multiple statements per line causes difficulty in reading the code. The measure is the number of continuations plus the number of multiple statement lines divided by the total number of lines for each module and then averaged over all of the modules in the system.

Execution Efficiency

EE.1 Performance Requirements allocated to design (design phase at system level). Performance requirements for the system must be broken down and allocated appropriately to the modules during the design. This metric simply identifies if the performance requirements have (1) or have not (0) been allocated during the design.

EE.2 Iterative Processing Efficiency Measure (design and implementation phases at module level first). The metric at the module level is the sum of the scores of the following applicable elements divided by the number of elements. At the system level it is an averaged score for all of the modules.

(1) Non-loop dependent computations kept out of loop.
Such practices as evaluating constants in a loop are to be avoided. This measure is based on the number of non-loop dependent statements

found in all loops in a module. This is to be measured from a detailed design representation during design and from the code during implementation.

(2) Performance Optimizing Compiler/Assembly language used (implementation only).
This is a binary measure which identifies if a performance optimizing compiler was used (1) or if assembly language was used to accomplish performance optimization (1) or not (0).

(3) Compound expressions defined once (implementation only).
Repeated compound expressions are to be avoided from an efficiency standpoint. This metric is based on the number of compound expressions which appear more than once.

(4) Number of overlays.
The use of overlays requires overhead as far as processing time. This measure, the inverse of the number of overlays, reflects that overhead. It can be applied during design when the overlay scheme is defined and during implementation.

(5) Free of bit/byte packing/unpacking in loops.
This is a binary measure indicating the overhead involved in bit/byte packing and unpacking. Placing these activities within loops should be avoided if possible.

(6) Module linkages (implementation only, requires execution).
This measure essentially represents the inter-module communication overhead. The measure is based on the amount of execution time spent during module to module communication.

(7) Operating System linkages (implementation only, requires execution).
This measure represents the module to OS communication overhead. The measure is based on the amount of execution time spent during module to OS communications.

(8) Efficient Use of storage facility.
This measure represents an evaluation of the utility of the storage facility.

EE.3 Data Usage Efficiency Measure (design and implementation phases applied at module level first). The metric at the module level is the sum of the scores of the following applicable elements divided by the number of applicable elements. The system metric is the averaged value of all of the module metric values.

(1) Data grouped for efficient processing.
The data utilized by any module is to be organized in the data base, buffers, arrays, etc., in a manner which facilitates efficient processing. The data organization during design and implementation is to be examined to provide this binary measure.

(2) Variables initialized when declared (implementation only).
This measure is based on the number of variables used in a module which are not initialized when declared.

Efficiency is lost when variables are initialized during execution of a function or repeatedly initialized during iterative processing.

(3) No mix-mode expressions (implementation only).
Processing overhead is consumed by mix-mode expressions which are otherwise unnecessary. This measure is based on the number of mix-mode expressions found in a module.

(4) Common choice of units/types.
For similar reasons as expressed above (3) this convention is to be followed. The measure is the inverse of the number of operations performed which have uncommon units or data types.

(5) Data indexed or referenced for efficient processing.
Not only the data organization, (1) above, but the linkage scheme between data items effects the processing efficiently. This is a binary measure of whether the indexing utilized for the data was chosen to facilitate processing.

Storage Efficiency

SE.1 Storage Efficiency Measure (design and implementation phases at module level first then system level). The metric at the module level is the sum of the scores of the following applicable elements divided by the number of applicable elements. The metric at the system level is the averaged value of all of the module metric values.

(1) Storage Requirements allocated to design (design phase only).
The storage requirements for the system are to be allocated to the individual modules during design. This measure is a binary measure of whether that allocation is explicitly made (1) or not (0).

(2) Virtual Storage Facilities Used.
The use of virtual storage or paging techniques enhances the storage efficiency of a system. This is a binary measure of whether these techniques are planned for and used (1) or not (0).

(3) Common data defined only once (implementation only).
Often, global data or data used commonly are defined more than once. This consumes storage. This measure is based on the number of variables that are defined in a module that have been defined elsewhere.

(4) Program Segmentation.
Efficient segmentation schemes minimize the maximum segment length to minimize the storage requirement. This measure is based on the maximum segment length. It is to be applied during design when estimates are available and during implementation.

(5) Dynamic memory management used.
This is a binary measure emphasizing the advantages of using dynamic memory management techniques to minimize the amount of storage required during execution. This is planned during design and used during implementation.

(6) Data packing used (implementation only).
While data packing was discouraged in EE.2 (5) in loops because of the overhead it adds to processing time, in general it is beneficial from a storage efficiency viewpoint. This binary measure applied during implementation recognizes this fact.

(7) Storage optimizing compiler/assembly language used (implementation only).

This binary measure is similar to EE.2 (2) except from the viewpoint of storage optimization.

Access Control

AC.1 Access Control Checklist (all three phases at system level).

The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) User I/O Access controls provided.

Requirements for user access control must be identified during the requirements phase. Provisions for identification and password checking must be designed and implemented to comply with the requirements. This binary measure applied at all three phases identifies whether attention has been placed on this area.

(2) Data Base Access controls provided.

This binary measure identifies whether requirements for data base controls have been specified, designed and the capabilities implementated. Examples of data base access controls are authorization tables and privacy locks.

(3) Memory protection across tasks.

Similar to (1) and (2) above, this measure identifies the progression from a requirements statement to implementation of memory protection across tasks. Examples of this type of protection, often times provided to some degree by the operating system, are preventing tasks from invoking other tasks, tasks from accessing system registers, and the use of privileged commands.

Access Audit

AA.1 Access Audit Checklist (all three phases at system level).

The metric is the averaged score of the following two elements.

(1) Provisions for recording and reporting access.

A statement of the requirement for this type capability must exist in the requirements specification. It is to be considered in the design specification, and coded during implementation. This binary metric applied at all three phases identifies whether these steps are being taken. Examples of the provisions which might be considered would be the recording of terminal linkages, data file accesses, and jobs run by user identification and time.

(2) Provisions for immediate indication of access violation.

In addition to (1) above, access audit capabilities required might include not only recording accesses but immediate identification of unauthorized accesses, whether intentional or not. This measure traces the requirement, design, and implementation of provisions for this capability.

Operability

OP.1 Operability Checklist (all three phases at system level).

The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) All steps of operation described.

This binary measure applied at all three phases identifies whether the operating characteristics have been described in the requirements specification, and if this description has been transferred into an implementable description of the operation (usually in an operator's manual). The description of the operation should cover the normal sequential steps and all alternative steps.

(2) All error conditions and responses appropriately described to operator.

The requirement for this capability must appear in the requirements specification, must be considered durinq design, and coded during implementation. Error conditions must be clearly identified by the system. Legal responses for all conditions are to be either documented and/or prompted by the system. This is a binary measure to trace the evolution and implementation of these capabilities.

(3) Provisions for operator to interrupt, obtain status, save, modify, and continue processing.

The capabilities provided to the operator must be considered during the requirements phase and then designed and implemented. Examples of operator capabilities include halt/resume and check pointing. This is a binary measure to trace the evolution of these capabilitie

(4) Number of operator actions reasonable (implementation only, requires execution).

The number of operator errors can be related directly to the number of actions required during a time period. This measure is based on the amount of time spent requiring manual operator actions divided by the total time required for the job.

(5) Job set up and tear down procedures described (implementation only). The specific tasks involved in setting up a job and completing it are to be described. This is usually documented during the implementation phase when the final version of the system is fixed. This is a binary measure of the existence of that description.

(6) Hard copy log of interactions maintained (design and implementation phases).
This is a capability that must be planned for in design and coded during implementation. It assists in correcting operational errors, improving efficiency of operation, etc. This measure identifies whether it is considered in the design and implementation phases (1) or not (0).

(7) Operator messages consistent and responses standard (design and implementation phases).
This is a binary measure applied during design and implementation to insure that the interactions between the operator and the system are simple and consistent. Operator responses such as YES, NO, GO, STOP, are concise, simple, and can be consistently used throughout a system. Lengthy, differently formated responses not only provide difficulty to the operator but also require complex error checking routines.

Training

TN.1 Training Checklist (design and implementation at system level). The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) Lesson Plans/Training Material developed for operators, end users, maintainers (implementation phase only).
This is a binary measure of whether this type documentation is provided during the implementation phase.

(2) Realistic simulated exercises provided (implementation only). This is a binary measure of whether exercises which represent the operational environment, are developed during the implementation phase for use in training.

(3) Sufficient 'help' and diagnostic information available on-line. This is a binary measure of whether the capability to aid the operator in familiarization with the system has been designed and built into the system. Provision of a list of legal commands or a list of the sequential steps involved in a process are examples.

Communicativeness

CM.1 User Input Interface Measure (all three phases at system level). The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) Default values defined (design and implementation). A method of minimizing the amount of input required is to provide defaults. This measure, applied during design and implementation, is based on the number of defaults allowed divided by the total number of input parameters.

(2) Input formats uniform (design and implementation). The greater the number of input formats there are the more difficult the system is to use. This measure is based on the total number of input formats.

(3) Each input record self-identifying. Input records which have self-identifying codes enhance the accuracy of user inputs. This measure is based on the number of input records that are not self identifying divided by the total number of input records. It is to be applied at design and implementation.

(4) Input can be verified by user prior to execution (design and implementation).
The capability, displaying input upon request or echoing the input automatically, enables the user to check his inputs before processing. This is a measure of the existence of the design and implementation of this capability.

(5) Input terminated by explicitly defined logical end of input (design and implementation).
The user should not have to provide a count of input cards. This is a binary measure of the design and implementation of this capability.

(6) Provision for specifying input from different media.
The flexibility of input must be decided during the requirements analysis phase and followed through during design and implementation. This is a binary measure of the existence of the consideration of this capability during all three phases.

CM.2 User Output Interface Measure (all three phases at system level).
The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) Selective Output Controls.
The existence of a requirement for, design for, and implementation of selective output controls is indicated by this binary measure. Selective controls include choosing specific outputs, output formats, amount of output, etc.

(2) Outputs have unique descriptive user oriented labels (design and implementation only).
This is a binary measure of the design and implementation of unique output labels. In addition, the labels are to be descriptive to the user. This includes not only the labels which are used to reference an output report but also the title, column headings, etc. within that report.

(3) Outputs have user oriented units (design and implementation). This is a binary measure which extends (2) above to the individual output items.

(4) Uniform output labels (design and implementation). This measure corresponds to CM.1 (2) above and is the inverse of the number of different output formats.

(5) Logical groups of output separated for user examination (design and implementation). Utilization of top of page, blank lines, lines of asterisks, etc., provide for easy identification of logically grouped output. This binary measure identifies if these techniques are used during design and implementation.

(6) Relationship between error messages and outputs is unambiguous (design and implementation). This is a binary measure applied during design and implementation which identifies if error messages will be directly related to the output.

(7) Provision for redirecting output to different media. This is a binary metric which identifies if consideration is given to the capability to redirect output to different media during requirements analysis, design, and implementation.

Software System Independence

SS.1 Software System Independence Measure (design and implementation phases at system level). The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) Dependence on Software System Utility programs.

The more utility programs, library routines, and other system facilities that are used within a system, the more dependent the system is on that software system environment. A SORT utility in one operating system is unlikely to be exactly similar to a SORT utility in another. This measure is based on the number of references to system facilities in a module divided by the total number of lines of code in the module. It is to be applied during design and implementation.

(2) Common, standard subset of language used.

The use of nonstandard constructs of a language that may be available from certain compilers cause conversion problems when the software is moved to a new software system environment. This measure represents that situation. It is based on the number of modules which are coded in a non-standard subset of the language. The standard subset of the language is to be established during design and adhered to during implementation.

Machine Independence

MI.1 Machine Independence Measure (design and implementation at system level). The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) Programming language used available on other machines. This is a binary measure identifying if the programming language used is available (1) on other machines or not (0). This means the same version and dialect of the language.

(2) Free from input/output references. Input and output references bind a module to the current machine configuration. Thus the fewer modules within a system that contain input and output references, the more localized the problem becomes when conversion is considered. This measure represents that fact and is based on the number of I/O references within a module. It is to be applied during design and implementation.

(3) Code is independent of word and character size (implementation only). Instructions or operations which are dependent on the word or character size of the machine are to be either avoided or parametric to facilitate use on another machine. This measure applied to the source during implementation is based on the number of modules which contain violations to the concept of independence of word and character size.

(4) Data representation machine independent (implementation only). The naming conventions (length) used are to be standard or compatible with other machines. This measure is based on the number of modules which contain variables which do not conform to standard data representations.

Communications Commonality

CC.1 Communications Commonality Checklist (all three phases at system level). The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) Definitive statement of requirements for communcation with other systems (requirements only).
During the requirement phase, the communication requirements with other systems must be considered. This is a binary measure of the existence of this consideration.

(2) Protocol standards established and followed.
The communcation protocol standards for communication with other systems are to be established during the design phase and followed during implementation. This binary measure applied at each of these phases indicates whether the standards were established and followed.

(3) Single module interface for input from another system.
The more modules which handle input the more difficult it is to interface with another system and implement standard protocols. This measure based on the inverse of the number of modules which handle input is to be applied to the design specification and source code.

(4) Single module interface for output to another system
For similar reasons as (3) above this measure is the inverse of the number of output modules.

Data Commonality

DC.1 Data Commonality Checklist (all three phases at system level). The metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

(1) Definitive statement for standard data representation for communications with other systems (requirements only).
This is a binary measure of the existence of consideration for standard data representation between systems which are to be interfaced. This must be addressed and measured in the requirements phase.

(2) Translation standards among representations established and followed (design and implementation).
More than one translation from the standard data representations used for interfacing with other systems may exist within a system. Standards for these translations are to be established and followed. This binary measure identifies if the standards are established during design and followed during implementation.

(3) Single module to perform each translation (design and implementation).
For similar reasons as CC.1 (3) and (4) above, this measure is the inverse of the maximum number of modules which perform a translation.

Conciseness

CO.1 Halstead's Measure (implementation phase at module level first then system level). The metric is based on Halstead's concept of length ([HALSM77]).

The observed length of a module is

$N_o = N_1 + N_2$ where:
N_1 = total usage of all operators in a module
N_2 = total usage of all operators in a module

The calculated length of a module is

$N_c = n_1 \log_2 n_1 + n_2 \log_2 n_2$ where:
n_1 = number of unique operators in a module
n_2 = number of unique operators in a module

The metric is normalized as follows:

$$1 - \frac{|N_c - N_o|}{N_o} \quad \text{or,}$$

$$0 \text{ if } \frac{|N_c - N_o|}{N_o} \text{ greater than } 1$$

At a system level the metric is the averaged value of all the module metric values.

REFERENCES

[ARRK74] Arrow, K. J.
"Limited Knowledge and Economic Analysis"
American Economic Review, March 1974.

[AR18-1] AR18-1 Management Information Systems Policies, Objectives, Procedures, and Responsibilities.

[BAUF73] Bauer, F. L. (Ed)
Advanced Course on Software Engineering
Springer - Verlag, Berlin, 1973.

[BOEB73] Boehm, B.
"Software and Its Impact: A Quantitative Report"
Datamation, April 1973.

[BOWJ78] Bower, J. L.
"The Business of Business is Serving Markets"
American Economic Review, May 1978, Vol. 68, No. 2.

[BRAR61] Brady, R. A.
Organization, Automation and Society: The Scientific Revolution in Industry, University of California Press, Berkeley and Los Angeles, 1961.

[CASM77] Cashman, M. W.
"An Interview With Professor Dr. Edsger W. Dijkstra"
Datamation, May 1977.

[CAVJ78] Cavano, J., McCall, J.
"A Framework for the Measurement of Software Quality", Proceedings ACM Software Quality Assurance Workshop, November 1978.

REFERENCE (continued)

[CHAR78] Chapman, R.
"Facing Financial Realities in Banking"
Datamation, June 1978.

[CHER] Chevance, R. J., et.al.
"Static Profile and Dynamic Behavior of COBOL Programs"
SIGPLAN, reference open.

[CONS75] Constantine, L., Yourdon, E.
"Structured Design", Yourdon Press, N. Y., 1975

[COTI75] Cotton, I. W.
"Microeconomics and the Market for Computer Services"
ACM Computing Surveys, Vol. 7, No. 2, June 1975.

[DeMR76] DeMillo, R. A., et.al.
"Can Structured Programs be Efficient?", SIGPLAN Notices, October 1976.

[DEWR78] Dewar, R., Hage, J.
"Size, Technology, Complexity, and Structual Differentiation: Toward a Theoretical Synthesis", Administrative Science Quarterly, pp 111-136, March 1978.

[DIJE69] Dijkstra, E. W.
NATO "Science Committee Report, January 1969".

[DIJE75] Dijkstra, E. W.
"Correctness Concerns and, Among Other Things, Why They are Resented", Proceedings of the 1975 International Conference on Reliable Software, Los Angeles.

REFERENCE (continued)

[DoDMAN] DoD Manual 4120.17-M
Automated Data Systems Documentation Standards

[DZIW78] Dzida, W., *et.al.*
"User-Perceived Quality of Interactive Systems", Proceedings of 3rd International Conference on Software Engineering.

[FAGM76] Fagan, M. E.
"Design and Code Inspections and Process Control in the Development of Programs", IBM Technical Report TR 00.2763, Poughkeepsie, 1976.

[FITA78] Fitzsimmons, A, Love, T.
"A Review and Evaluation of Software Science", ACM *Computing Surveys*, Vol. 10, No. 1, March 1978.

[FLEJ72] Fleiss, J. E., *et.al.*
"Programming for Transferability"
NTIS Memorandum AD-750 897, 1972.

[FOSL76] Fosdick, L. D., Osterweil, L. J.
"Data Flow Analysis in Software Reliability", ACM *Computing Surveys* Special Issue: Reliable Software I, 1976.

[FRIR78] Fried, R.
"Monitoring Data Integrity"
Datamation, June 1978.

[FAIE78] Gainer, E., *et.al.*
"The Design of a Reliable Applications System"
3rd *Proceedings*.

REFERENCE (continued)

[GALJ73] Galbraith, J.
Designing Complex Organizations
Addison-Wesley, Reading, Mass 1973.

[GETC78] Getz, C. W.
"DP's Role is Changing"
Datamation, February 1978.

[GOLJ73] Goldberg, J., ed.
Proceedings of the Symposium on the High Cost of Software,
Monterey, 1973.

[GORG71] Gorry, G. A., Scott Morton, M.S.
"A Framework for Management Information Systems"
Sloan Management Review, Vol. 13, No. 1,
Fall 1971, MIT Cambridge, Mass.

[HANS76] Hantler, S. L., King, J. C.
"An Introduction to Proving the Correctness of Programs"
ACM Computing Surveys Special Issue: Reliable Software I,
September 1976.

[HECS77] Hecht, M. S.
Flow Analysis of Computer Programs, Elsevier North-Holland,
New York, 1977.

[HETB78] Hetzel, B.
"A Perspective on Software Development"
3rd Proceedings.

REFERENCES (continued)

[HIBP78] Hibbard, P. G., Schuman, S. A.
Constructing Quality Software, IFIP Working Conference,
Novosibirsk, USSR North Holland 78

[HIRA70] Hirschman, A. O.
Exit, Voice, and Loyalty: Responses to Decline in Firms,
Organizations and States
Harvard University Press, Cambridge, Mass 1970.

[HOAC78] Hoare, C.A.R.
"Software Engineering: A Keynote Address", 3rd Proceedings
of the International Conference on Software Engineering,
Atlanta, 1978.

[HOLJ77] Holton, J. B.
"Are the New Programming Techniques Being Used?"
Datamation, July 1977.

[HORJ73] Horning, J. J., Randell, B.
"Process Structuring"
ACM Computing Surveys, Vol. 5, No. 1, March 1973.

[JACM78] Jackson, M. A.
"Information Systems: Modeling, Sequencing and Transformations"
3rd Proceedings of the International Conference on Software
Engineering, Atlanta, 1978.

[KEEP77] Keen, P. G. W., Gerson, E. M.
"The Politics of Software Systems Design"
Datamation, November 1977.

REFERENCES (continued)

[KINJ78] King, J. L., Schrems, E. L.
"Cost-Benefit Analysis in Information Systems Development and Operation", ACM Computing Surveys, Vol. 10, No. 1, March 1978.

[KLEL76] Klein, L.
New Forms of Work Organization
Cambridge University Press, Cambridge, 1976.

[KLIR78] Kling, R.
"Value Conflicts and Social Choice in Electronic Funds Transfer System Developments", CACM, Vol. 21, No. 8, August 1978.

[KNUD73] Knuth, D. E.
"A Review of "Structured Programming"", Computer Science Dept. STAN-CS-73-371, Stanford University.

[KOSS74] Kosaraju, S. R., Ledgard, M. F.
Concepts in Quality Software Design
NBS Technical Note 842, Washington 1974.

[KURS75] Kurki-Suonio, R.
"Towards Better Structured Definitions of Programming Languages", STAN-CS-75-500 Computer Science Dept., Stanford University, 1975.

[LEIH78] Leibenstein, H.
"On the Basic Proposition of X-Inefficiency Theory"
American Economic Review, May 1978.

REFERENCES (continued)

[LINT76] Linden, T. A.
"Operating System Structures to Support Security and Reliable Software", ACM Computing Surveys, Vol. 8, No. 4, 1976.

[LITB78] Littlewood, B.
"How to Measure Reliability, and How Not to..."
3rd Proceedings of the International Conference on Software Engineering, Atlanta, 1978.

[LOVL77] Love, L. T.
Relating Individual Differences in Computer Programming Performance to Human Information Processing Abilities,
Ph.D Thesis University of Washington, 1977.

[LOVT77a] Love, T.
An Experimental Investigation of the Effect of Program Structure on Program Understanding,
G.E. Technical Information Series TIS77ISP006.

[LOVT77b] Love, T.
A Preliminary Experiment to Test Influences on Human Understanding of Software,
G. E. Technical Information Series TIS77ISP007.

[LUCH74] Lucas, H. C.
Toward Creative Systems Design
Solumbia University Press, New York 1974.

[LYOG78] Lyon, G.
"COBOL Instrumentation and Debugging: A Case Study"
NBS Special Publication 500-26, U. S. Dept. of Commerce 1978.

REFERENCES (continued)

[MARR71] Marris, R., Wood, A., eds.
The Corporate Economy Growth, Competition and Innovative Potential, Harvard University Press, Cambridge, Mass, 1971.

[MATM78] Matsumoto, M.
"Design and Quality in MIS Environments"
Software Metrics Enhancement Task Internal Memorandum No. 1, August 1978.

[McCC78] McClure, C. L.
Reducing COBOL Complexity through Structured Programming
Van Nostrand Reinhold Co., 1978.

[McCJ77a] McCall, J., Richards, P., Walters, G.
"Factors in Software Quality", 3 Vols. (A049014) (A049015) (A049055)
RADC TR 77-369, November 1977

[McCJ77b] McCall, J., Richards, P., Walters, G.
"Metrics for Software Quality Evaluation and Prediction"
Proceedings of the NASA/Goddard Second Summer Engineering Workshop, September 1977.

[McCJ78a] McCall, J.
"The Utility of Software Quality Metrics in Large-Scale Software System velopments", Proceedings of the Second Software Life Cycle Management Workshop, August 1978.

[McCJ78b] McCall, J.
"Software Quality: The Illusive Measurement"
Software Quality Management Conference, September 1978.

REFERENCES (continued)

[McCP78] McCarter, P. M.
"Where is the Industry Going?"
Datamation, February 1978.

[McCK75] McKeeman, W. M.
"On Preventing Programming Languages from Interfering with Programming", IEEE Transactions on Software Engineering, March 1975.

[MILSTD] MIL-STD-490
Specification Practices

[MIYI78] Miyamoto, I.
"Toward an Effective Software Reliability Evaluation"
3rd Proceedings of the International Conference on Software Engineering, Atlanta, 1978.

[MYEG75] MYERS, G. S.
Reliable Software Through Composite Design
Petrocelli/Charter, 1975.

[NOBD77] Noble, D.
America By Design, Harper Row,
New York, 1977.

[PAND76] Panzl, D. J.
"Test Procedures: A New Approach to Software Verification"
Proceedings of the Second International Conference on Software Engineering, San Francisco 1976.

REFERENCES (continued)

[PARE75] Parker, E. B.
"Social Implications of Computer/Telecommunications Systems",
Conference on Computer/Telecommunications Policy - Organization for Economic Co-operation and Development
Paris, 4-6 February 1975.

[PARD75] Parnas, D. L.
"The Influence of Software Structure on Reliability
Proceedings of the International Conference on Reliable Software, Los Angeles, 1975.

[PEDJ78] Pederson, J. T., Buckle, J. K.
"Kongsberg's Road to an Industrial Software Methodology"
3rd Proceedings.

[PEED78] Peeples, D. E.
"Measure for Productivity"
Datamation, May 1978.

[PETJ77] Peterson, J. L.
"Petri Nets", ACM Computing Surveys, Vol. 9, No. 3, 1977

[PODJ77] Podolsky, J. L.
"Horace Builds a Life Cycle",
Datamation, November 1977.

[PRO73] "Proceedings of a Symposium on the High Cost of Software"
AFOSR, ARO, ONR, 1973.

[PRO75] "Proceedings of the International Conference on Reliable Software", ACM, 1975.

REFERENCES (continued)

[PYSA78] Pyster, A., Dutra, A.
"Error-Checking Compilers and Portability"
Software Practice and Experience, Vol. 8, Issue 1,
January - February 1978.

[RICP76] Richards, P., Chang, P.
"Localization of Variables: A Measure of Complexity"
GE TIS 76CIS07, December 1976.

[RICD70] Richardson, D. W.
Electric Money: Evolution of an Electronic Funds -
Transfer System, MIT Press, Cambridge, Mass, 1970.

[RIDW78] Riddle, W. E., et.al.
"Behavior Modelling During Software Design"
3rd Proceedings of the International Conference on Software
Engineering, Atlanta, 1978.

[ROBL75] Robinson, L., et.al.
"The Verification of COBOL Programs"
NTIS Memorandum, June 1975.

[ROGE76] Rogers, E. M., Agarwala-Rogers, R.
Communications in Organizations
The Free Press, New York, 1976.

[SAMS76] "Contractor Software Quality Assurance Evaluation Guide"
SAMSO Pamphlet 74-2, Los Angeles, 1976.

[SCOM] Scott Morton, M.S.
"Some Perspectives on Computerized Management Decision
Making Systems", Unpublished draft

REFERENCES (continued)

[SHAW69] Sharpe, W. F.
The Economics of Computers
Columbia University Press, New York, 1969.

[SPEA74] Spence, A. M.
"An Economist's View of Information"
Annual Review of Information Science, Vol. 9, 1974.

[STAR73] Stamper, R.
Information in Business and Administrative Systems
John Wiley and Sons, New York, 1973.

[STR74] "Structured Programming Series"
RADC, 15 Vols., 1974-1975.

[TAGW77] Taggart, W. M. Jr, Tharp, M. O.
"A Survey of Information Requirements Analysis Techniques"
ACM Computing Surveys, Vol. 9, No. 4, 1977.

[THOD78] Thomas, D. R. E.
"Strategy is Different in Service Businesses"
Harvard Business Review, July-August 1978
pp 158-165, Cambridge, Mass.

[USACSCM] USACSC Manual 18-1
Automatic Data Processing System Development, Maintenance and Documentation Standards and Procedures Manual.

[VINW77] Vinson, W. D., Heany, D. F.
"Is Quality Out of Control?"
Harvard Business Review, November-December 1977.

REFERENCES (continued)

[WALG78a] Walters, G., McCall, J.
"The Development of Metrics for Software R&D"
1978 Proceedings, Annual Reliability and Maintainability Symposium, January 1978.

[WALG78b] Walters, G.
"Application of Metrics to Software Quality Management Programs", Software Quality Management Conference, September 1978.

[WEGP76] Wegner, P.
"Research Paradigms in Computer Science"
Proceedings of the 2nd International Conference on Software Engineering, San Francisco, 1976.

[WEGP78] Wegner, P.
"Research Directions in Software Technology"
3rd Proceedings.

[WIRN65] With, N.
"On Certain Basic Concepts of Programming Languages"
Technical Report No. CS65, Computer Science Department, Stanford University, 1965.

[WONG78] Wong, G.
"Design Methodology for Computer System Modeling Tools"
Symposium on Modeling and Simulation Methodology, August 1978, Rehorot, Isreal.

[YEHR76] Yeh, R. T., ed.
ACM Computing Surveys Special Issue! Reliable Software I: Software Validation 1976.

REFERENCES (Continued)

[CULK79] Culik
"The Cyclomatic Number and the Normal Number of Programs"
ACM SIGPLAN Notices, Vol. 14, No. 4, April 1979

[MILE79] Miller, E.
"Some Statistics from the Software Test Factory"
ACM Software Engineering Notes, Vol. 4, No.1, January 1979.

[JOHJ75] Johnson, J.P.
"Software Reliability Measurement"
NTIS AD-A019-147, December 1975.

[KAUR75] Kauffman, R.
"COBOL/Structured Programming - Win the Marriage Survive"
Infosystems, February 1975.

[GELD79] Gelperin, D.
"Testing Maintainability"
ACM Software Engineering Notes, Vol. 4, No. 2, April 1979.

[ALJM79] Al-Jarrah, M., et. al.
"An Empirical Analysis of COBOL Programs"
Software - Practice and Experience, Vol. 9, Issue No. 5,
May 1979.

[MCKJ79] McKissick, J., et. al.
"The Software Development Notebook - A Proven Technique"
Proceedings 1979 Annual Reliability and Maintainability Symposium, January 1979

[IMP74] "Improved Programming Technologies - An Overview"
IBM TR-GC20-1850-0, 1974.

REFERENCES (Continued)

[LIEB78] Lientz, B., et. al.
"Characteristics of Applications Software Maintenance"
Communications of the ACM, Vol. 21, No. 6, June 1978.

[BASV78] Basil, V., et. al.
"Investigating Software Development Approaches"
AFOSR TR-688, August 1978.

[PHIM76] Phister, M.
Data Processing Technology and Economics, Santa Monica Publishing Co., 1976.

MISSION
Of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C³I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.