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

SOFTWARE QUALITY MEASUREMENT MANUAL

General Electric Company

James A. McCall
Mike T. Matsumoto

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

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

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 II (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.

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 (C^3I) 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.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-80-109 - Vol II (of two)	2. GOVT ACCESSION NO.: AD-A086986
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): SOFTWARE QUALITY MEASUREMENT MANUAL. Volume II.	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, Jun[redacted]78 - Jul[redacted]979
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): James A. McCall, Mike T. Matsumoto	8. CONTRACT OR GRANT NUMBER(s): F30602-78-C-0216
9. PERFORMING ORGANIZATION NAME AND ADDRESS: General Electric Company, Information Systems Programs, 450 Persian Drive, Sunnyvale CA 94086	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 63728F, 2528[redacted]
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIS), Griffiss AFB NY 13441	12. REPORT DATE: Apr[redacted]80
	13. NUMBER OF PAGES: 67
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	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.

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

Same

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.

(Cont'd)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

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.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION 1

INTRODUCTION

1.1 PURPOSE

There has been an increased awareness in recent years of the critical problems that have been encountered in the development of large scale software systems. These problems not only include the cost and schedule overruns typical of development efforts, and the poor performance of the systems once they are delivered, but also include the high cost of maintaining the systems, the lack of portability, and the high sensitivity to changes in requirements.

The government and DOD in particular, as customers of many large scale software system developments, have sponsored many research efforts aimed at attacking these problems. For example, the efforts related to the development of a standard DOD programming language, software development techniques, and development tools and aids all provide partial solution to the above problems by encouraging a more disciplined approach to the development of software and therefore a more controlled development process.

A related research thrust which has been recently funded by DOD is the area of software metrics. The research in this area has resulted in the development and evaluation of a number of metrics which measure various attributes of software and relate to different aspects of software quality.

The potential of the software metric concepts can be realized by their inclusion in software quality assurance programs. Their impact on a quality assurance program is to provide a more disciplined, engineering approach to quality assurance and to provide a mechanism for taking a life cycle viewpoint of software quality. The benefits derived from their application are realized in life cycle cost reduction.

The purpose of this manual is to present a complete set of procedures and guidelines for introducing and utilizing current software quality measurement techniques in a quality assurance program associated with large scale software system developments. These procedures and guidelines will identify:

1. How to identify and specify software quality requirements (Setting Quality Goals).

2. How and when to apply software metrics (Applying Metrics), and

3. How to interpret the information obtained from the application of the metrics (Making a Quality Assessment).

1.2 SCOPE

This manual is based on the results of research conducted in support of the United States Air Force Electronic Systems Division (ESD), Rome Air Development Center (RADC), and the United States Army Computer Systems Command's Army Institute for Research in Management Information and Computer Science (USACSC/AIRMICS). While aspects of the technology of software metrics require further research, those portions which can currently provide benefit to a software quality assurance program are emphasized in this manual. Guidelines and procedures for using the software metrics are described. The guidelines and procedures are presented in such a way as to facilitate their application using this manual in a software development. All of the procedures are described as manual processes. however, where automated software tools could be used to compliment or enhance the process, the tools are identified.

1.3 QUALITY MEASUREMENT IN PERSPECTIVE

The evolution during the past decade of modern programming practices, structured, disciplined development techniques and methodologies, and requirements for more structured, effective documentation, has increased the feasibility of effective measurement of software quality.

However, before the potential of measurement techniques could be realized a framework or model of software quality had to be constructed. An established model, which at one level provides a user or management oriented view of quality, is described in Section 2 of this manual in the perspective of how it can be used to establish software quality requirements for a specific application.

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, strucure, density, complexity type attributes. They provide an indication of the quality that will be achieved in the end product, based on the nature of the 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 described and used in this manual 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, 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.

A summarization of how the software metric concepts complement Quality Assurance activities is provided in Table 1.3-1 based on the quality assurance program requirements identified in MIL-S-52779.

Table 1.3-1 How Software Metrics Complement Quality Assurance

QUALITY ASSURANCE PROGRAM REQUIREMENTS	IMPACT OF SOFTWARE QUALITY METRIC CONCEPTS
• Assure Conformance with Requirements	Adds software quality requirements
• Identify Software Deficiencies	Anomaly-detecting metrics
• Provide Configuration Management	No impact
• Conduct Test	Assists in evaluation of other qualities
• Provide Library Controls	No impact
• Review Computer Program Design	Predictive metrics
• Assure Software Documentation Requirement Conformance	Metrics assist in evaluation of documentation as well as code
• Conduct Reviews and Audits	Procedures for applying metrics (in form of worksheets) formalizes inspection process
• Provide Tools/Techniques/ Methodology for Quality Assurance	This manual describes methodology of using metrics
• Provide Subcontractor Control	No impact

All of these concepts will be further explained and illustrated in the subsequent sections of this manual.

1.4 MANUAL ORGANIZATION

The manual has been organized as a handbook for use in a quality assurance program. The first section provides introductory information and how the manual is to be used.

The second section defines the software quality model and describes a methodology for using this model to establish software quality requirements or goals for a software development.

The third section describes procedures for measuring the quality of the software. These procedures cover what to measure, when to measure, and how to measure.

The fourth section describes procedures for utilizing the information provided by the measurements to make assessments of the quality of the software and recommends what information to present to various personnel involved in the development.

1.5 RECOMMENDED USE OF MANUAL

The software quality metric concepts can be applied at several levels. In an acquisition manager/contractor environment, there are three approaches for using the metric concepts. They are:

1. The acquisition manager's staff can apply metrics to the delivered software procucts.

2. The development manager's staff can apply metrics to software products and report them to the acquisition manager during reviews.

3. An independent Quality Assurance contractor can apply metrics to delivered software products and report them to the acquisition manager.

Within the software development project organization, there are two approaches for using the metric concepts. They are:

1. The quality assurance personnel will apply the metrics as an independent assessment of the quality of the software being produced.

2. The development personnel can apply the metrics during walkthroughs and reviews.

This manual is oriented toward those personnel who will be applying the concepts (either quality assurance or development personnel) and recommends three approaches to both establishing the quality requirements (Section 2) and making a quality assessment (Section 4). The three approaches (an index is provided in Table 1.5-1) in each area are presented in order of increasing formality of the relationship between quality requirements and the metrics, that is in order of increasing quantification. The order of presentation also relates to an increasing requirement for experience with the concepts by the personnel applying the concepts. Thus, the approaches can be used as a phased implementation plan for the metric concepts. It is recommended that the concepts be incrementally phased into the quality assurance organization's operation.

This manual should be utilized by the personnel applying the metric concepts. Additional information and definitions can be found in:

"Factors in Software Quality", 3 vols, RADC-TR-77-369, Nov 1977.[McCA77]

"Software Quality Metrics Enhancements-Final Report", Vol I of this document

These references should be read by the personnel applying the metrics to familiarize them with the underlying concepts. They should also be referred to periodically for definitions and explanation purposes.

Table 1.5-1 Index of Three Approaches to Specifying and Assessing Software Quality

APPROACH (LEVEL OF FORMALITY)	SPECIFYING SOFTWARE QUALITY	APPLYING MEASUREMENTS	ASSESSING THE QUALITY OF THE PRODUCT
1	Procedures for identifying important quality factors (Paragraph 2.2)	PROCEDURES FOR APPLYING THE METRIC WORKSHEETS (SECTION 3)	Procedures for the inspector's assessment (Paragraph 4.2)
2	Procedures for identifying critical software attributes (Paragraph 2.3)		Procedures for performing sensitivity analysis (Paragraph 4.3)
3	Procedures for identifying quantifiable goals (Paragraph 2.4)		Procedures for use of normalization function (Paragraph 4.4)

SECTION 2
PROCEDURES FOR
INDENTIFYING SOFTWARE QUALITY REQUIREMENTS

2.1 INTRODUCTION

The primary purpose of applying Software Quality Metrics in a Quality Assurance Program is to improve the quality of the software product. Rather than simply measuring, the concepts are based on achieving a positive influence on the product, to improve its development.

This section addresses the problem of identifying software quality requirements or goals. These requirements are in addition to the functional, performance, cost, and schedule requirements normally specified for a software development. The fact that the goals established are related to the quality of the end product should, in itself, provide some positive influence. Past research has shown that goal-directed system development is effective. [WEIN72]

The vehicle for establishing the requirements is the hierarchical model of software quality defined in [CAVA78]. This model, shown in Figure 2.1-1, has at its highest level a set of software quality factors which are user/management-oriented terms and represent the characteristics which comprise software quality. At the next level for each quality factor, is a set of criteria which are the attributes if present, that provide the characteristics represented by the quality factors. The criteria, then, are software-related terms. At the lowest level of the model are the metrics which are quantitative measures of the software attributes defined by the criteria.

The procedures for establishing the quality requirements for a particular software system utilize this model and will be described as a three level approach, the levels corresponding to the hierachical levels of the software quality model. The first level establishes the quality factors that are important. The second level identifies the critical software attributes. The third level identifies the metrics which will be applied and establishes quantitative ratings for the quality factors.

Once the quality requirements have been determined by following the procedures described in the subsequent paragraphs, they must be transmitted to the development team. In a formal acquisition manager/contractor environment, the Request for Proposal (RFP) is the medium for identifying these requirements. The results of following the procedures should be incorporated in the RFP. If the development is being done internally, the quality requirements should be documented in the same form as the other system requirements and provided to the development team. Additionally, a briefing emphasizing the intent of the inclusion of the quality requirements is recommended.

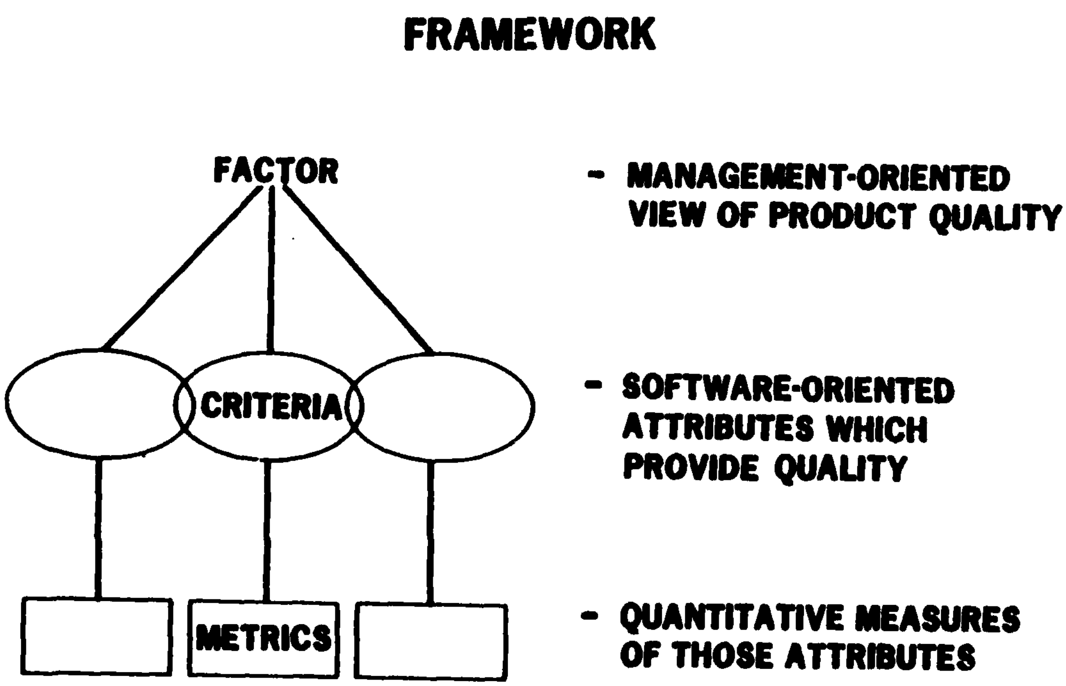


Figure 2.1-1

2.2 PROCEDURES FOR IDENTIFYING IMPORTANT QUALITY FACTORS

2.2.1 PROCEDURES

The basic tool to be utilized in identifying the important quality factors will be the Software Quality Requirements Survey form shown in Table 2.2-1. The formal definitions of each of the eleven quality factors are provided on that form.

It is recommended that a briefing be provided to the decision makers using the tables and figures which follow in this paragraph to solicit their responses to the survey. The decision makers may include the acquisition manager, the user/customer, the development manager, and the QA manager. To complete the survey the following procedures should be followed:

1a. *Consider Basic Characteristics of the Application*

The software quality requirements for each system are unique and are influenced by system or application-dependent characteristics. There are basic characteristics which affect the quality requirements and each software system must be evaluated for its basic characteristics. Table 2.2-2 provides a list of some of these basic characteristics. For example, if the system is being developed in an environment in which there is a high rate of technical breakthroughs in hardware design, Portability should take on an added significance. If the expected life cycle of the system is long, Maintainability becomes a cost-critical consideration. If the application is an experimental system where the software specifications will have a high rate of change, Flexibility in the software product is highly desirable. If the functions of the system are expected to be required for a long time, while the system itself may change considerably, Resuability is of prime importance in those modules which implement the major functions of the system. With the advent of more computer networks and communication capabilities, more systems are being required to interface with other systems

Table 2.2-1 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)______

and the concept of Interoperability is extremely important. These and other system characteristics should be considered when identifying the important quality factors.

Table 2.2-2 System Characteristics and Related Quality Factors

CHARACTERISTIC	QUALITY FACTOR
• If human lives are affected	Reliability Correctness Testability
• Long life cycle	Maintainability Flexibility Portability
• Real time application	Efficiency Reliability Correctness
• On-board computer application	Efficiency Reliability Correctness
• Processes classified information	Integrity
• Interrelated systems	Interoperability

1b. *Consider Life Cycle Implications*

The eleven quality factors identified on the survey can be grouped according to three life cycle activities associated with a delivered software product. These three activities are product operation, product revision, and product transition. The relationship of the quality factors to these activities is shown in Table 2.2-3. This table also illustrates where quality indications can be achieved through measurement and where the impact is felt if poor quality is realized. The size of this impact determines the cost savings that can be expected if a higher quality system is achieved through the application of the metrics. This cost savings is somewhat offset by the cost to apply the metrics and the cost to develop the higher quality software product as illustrated in Figure 2.2-1.

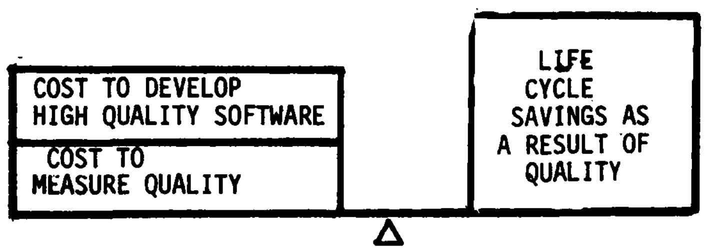


Figure 2.2-1
Cost vs Benefit Tradeoff

This cost to implement versus life cycle cost reduction relationship exists for each quality factor. The benefit versus cost-to-provide ratio for each factor is rated as high, medium, or low in the right hand column of Table 2.2-3. This relationship and the life cycle implications of the quality factors should be considered when selecting the important factors for a specific system.

Table 2.2-3 The Impact of Not Specifying or Measuring Software Quality Factors

LIFE-CYCLE PHASES / FACTORS	DEVELOPMENT			EVALUATION	POST-DEVELOPMENT			EXPECTED COST SAVED / COST TO PROVIDE
	REQMIS ANALYSIS	DESIGN	CODE & DEBUG	SYSTEM TEST	OPERATION	REVISION	TRANSITION	
CORRECTNESS	△	△	△	X	X	X		HIGH
RELIABILITY	△	△	△	X	X	X		HIGH
EFFICIENCY	△	△	△		X			LOW
INTEGRITY	△	△	△		X			LOW
USABILITY	△	△		X		X		MEDIUM
MAINTAINABILITY		△	△			X	X	HIGH
TESTABILITY		△	△	X		X	X	HIGH
FLEXIBILITY		△	△			X	X	MEDIUM
PORTABILITY		△	△				X	MEDIUM
REUSABILITY		△	△				X	MEDIUM
INTEROPERABILITY	△	△		X			X	LOW

LEGEND: - where quality factors should be measured X - where impact of poor quality is realized

1c. *Perform Tradeoffs Among the Tentative List of Quality Factors.* As a result of steps 1a and 1b, a tentative list of quality factors should be produced. The next step is to consider the interrelationships among the factors selected. Table 2.2-4 and 2.2-5 can be used as a quide for determing the relationships between the quality factors. Some factors are synergistic while others conflict. The impact of conflicting factors is that the cost to implement will increase. This will lower the benefit to cost ratio described in the preceeding paragraph.

Table 2.2-4 Relationships Between Software Quality Factors

FACTORS	CORRECTNESS	RELIABILITY	EFFICIENCY	INTEGRITY	USABILITY	MAINTAINABILITY	TESTABILITY	FLEXIBILITY	PORTABILITY	REUSABILITY	INTEROPERABILITY
CORRECTNESS											
RELIABILITY	○										
EFFICIENCY											
INTEGRITY			●								
USABILITY	○	○	●	○							
MAINTAINABILITY	○	○	●		○						
TESTABILITY	○	○	●		○	○					
FLEXIBILITY	○	○	●	●	○	○	○				
PORTABILITY			●			○	○				
REUSABILITY		●	●	●		○	○	○	○		
INTEROPERABILITY			●	●					○		

LEGEND

If a high degree of quality is present for factor, what degree of quality is expected for the other:

○ = High ● = Low

Blank = No relationship or application dependent

Table 2.2-5 Typical Factor Tradeoffs

INTEGRITY VS EFFICIENCY	The additional code and processing required to control the access of the software or data usually lengthens run time and require additional storage.
USABILITY VS EFFICIENCY	The additional code and processing required to ease an operator's tasks or provide more usable output usually lenghten run time and require additional storage.
MAINTAINABILITY VS EFFICIENCY	Optimized code, incorporating intricate coding techniques and direct code, always provides problems to the maintainer. Using modularity, instrumentation, and well commented high level code to increase the maintainability of a system usually increases the overhead resulting in less efficient operation.
TESTABILITY VS EFFICIENCY	The above discussion applies to testing.
PORTABILITY VS EFFICIENCY	The use of direct code or optimized system software or utilities decreases the portability of the system.
FLEXIBILITY VS EFFICIENCY	The generality required for a flexible system increases overhead and decreases the efficiency of the system.
REUSABILITY VS EFFICIENCY	The above discussion applies to reusability.
INTEROPERABILITY VS EFFICIENCY	Again the added overhead for conversion from standard data representations, and the use of interface routines decreases the operating efficiency of the system.
FLEXIBILITY VS INTEGRITY	Flexibility requires very general and flexible data structures. This increases the data security problem.
REUSABILITY VS INTEGRITY	As in the above discussion, the generality required by reusable software provides severe protection problems.
INTEROPERABILITY VS INTEGRITY	Coupled systems allow for more avenues of access and different users who can access the system. The potential for accidental access of sensitive data is increased as well as the opportunities for deliberate access. Often, coupled systems share data or software which compounds the security problems as well.
REUSABILITY VS RELIABILITY	The generality required by reusable software makes providing error tolerance and accuracy for all cases more difficult.

1d. *Identify Most Important Quality Factors*

Based on 1a through 1c, a list of quality factors considered to be important for the particular system should be compiled. The list should be organized in order of importance. A single decision maker may choose the factors or the choice may be made by averaging several survey responses. The definitions of the factors chosen should be included with this list.

1e. *Provide Explanation for Choice*

The rationale for the decisions made during steps 1a through 1c should be documented.

2.2.2 AN EXAMPLE OF FACTORS SPECIFICATION

To illustrate the application of these steps, consider a tactical inventory control system. The inventory control system maintains inventory status and facilitates requistitioning, reordering, and issuing of supplies to Army units in combat situations. The planned life of the system is ten years.

Each step described previously will be performed with respect to the tactical inventory control system.

1a. *Consider Basic Characteristics of the Application*

Utilizing table 2.2-2 and considering the unique characteristics of the tactical inventory control system resulted in the following:

Characteristic	Related Quality Factor
Critical Support for Combat Units	*Reliability* *Correctness*
Long Life Cycle With Stable Hardware And Software Requirements	*Maintainability*
Utilized By Army Supply Personnel	*Usability*

Interfaces with Inventory Systems At Other Army Echelons	*Interoperability*

1b. *Consider Life Cycle Implications*

Of the five quality factors identified in 1a, all provide high or medium life cycle cost benefits according to Table 2.2-3.

FACTORS	COST BENEFIT RATIO
Reliablity	*High*
Correctness	*High*
Maintainability	*High*
Usability	*Medium*
Interoperability	*High*

1c. *Perform Trade Offs Among Factors*

Using Table 2.2-4, there are no conflicts which need to be considered.

1d. *Identify Most Important Quality Factors*

Using the survey form, Table 2.2-1, and the guidance provided by steps 1a through 1c, the following factors are identified in order of importance. The definitions are provided.

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.
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.
INTEROPERABILITY	-Effort required to couple one system with another.

1e. *Provide Explanation for Choice*

CORRECTNESS	-System performs critical supply function
RELIABILITY	-System performs critical supply functions in field environment
USABILITY	-System will be used by military personnel with minimum computer training
MAINTAINABILITY	-System life cycle is projected to be 10 years and will operate in the field where military personnel will maintain.
INTEROPERABILITY	-System will interface with other supply systems at higher levels of command

2.3 PROCEDURES FOR IDENTIFYING CRITICAL SOFTWARE ATTRIBUTES

2.3.1 PROCEDURES

The next level of identifying the quality requirements involves proceeding from the user-oriented quality factors to the software-oriented criteria. Sets of criteria, which are attributes of the software, are related to the various factors by definition. Their identification is automatic and represents a more detailed specification of the quality requirements.

2a. *Identify Critical Software Attributes Required*

Table 2.3-1 should be used to identify the software attributes associated with the chosen critical quality factors.

Table 2.3-1 Software Criteria and Related Quality Factors

FACTOR	SOFTWARE CRITERIA
CORRECTNESS	TRACEABILITY CONSISTENCY COMPLETENESS
RELIABILITY	ERROR TOLERANCE CONSISTENCY ACCURACY SIMPLICITY
EFFICIENCY	STORAGE EFFICIENCY EXECUTION EFFICIENCY
INTEGRITY	ACCESS CONTROL ACCESS AUDIT
USABILITY	OPERABILITY TRAINING COMMUNICATIVENESS
MAINTAINABILITY	CONSISTENCY

FACTOR	SOFTWARE CRITERIA
FLEXIBILITY	MODULARITY GENERALITY EXPANDABILITY
TESTABILITY	SIMPLICITY MODULARITY INSTRUMENTATION SELF-DESCRIPTIVENESS
PORTABILITY	MODULARITY SELF-DESCRIPTIVENESS MACHINE INDEPENDENCE SOFTWARE SYSTEM INDEPENDENCE
REUSABILITY	GENERALITY MODULARITY SOFTWARE SYSTEM INDEPENDENCE
INTEROPERABILITY	MODULARITY COMMUNICATION COMMONALITY DATA COMMONALITY

2b. *Provide Definitions*

The definitions in Table 2.3-2 should also be provided as part of the specification.

Table 2.3-2 Criteria Definitions for Software Quality Factors

CRITERION	DEFINITION
TRACEABILITY	Those attributes of the software that provide a thread from the requirements to the implementation with respect to the specific development and operational environment.
COMPLETENESS	Those attributes of the software that provide full implementation of the functions required.
CONSISTENCY	Those attributes of the software that provide uniform design and implementation techniques and notation.
ACCURACY	Those attributes of the software that provide the required precision in calculations and outputs.
ERROR TOLERANCE	Those attributes of the software that provide continuity of operation under nonnominal conditions.
SIMPLICITY	Those attributes of the software that provide implementation of functions in the most understandable manner. (Usually avoidance of practices which increase complexity.)
MODULARITY	Those attributes of the software that provide a structure of highly independent modules.
GENERALITY	Those attributes of the software that provide breadth to the functions performed.
EXPANDABILITY	Those attributes of the software that provide for expansion of data storage requirements or computational functions.
INSTRUMENTATION	Those attributes of the software that provide for the measurement of usage or identification of errors.
SELF-DESCRIPTIVENESS	Those attributes of the software that provide explanation of the implementation of a function.

Table 2.3-2 Criteria Definitions for Software Quality Factors (Continued)

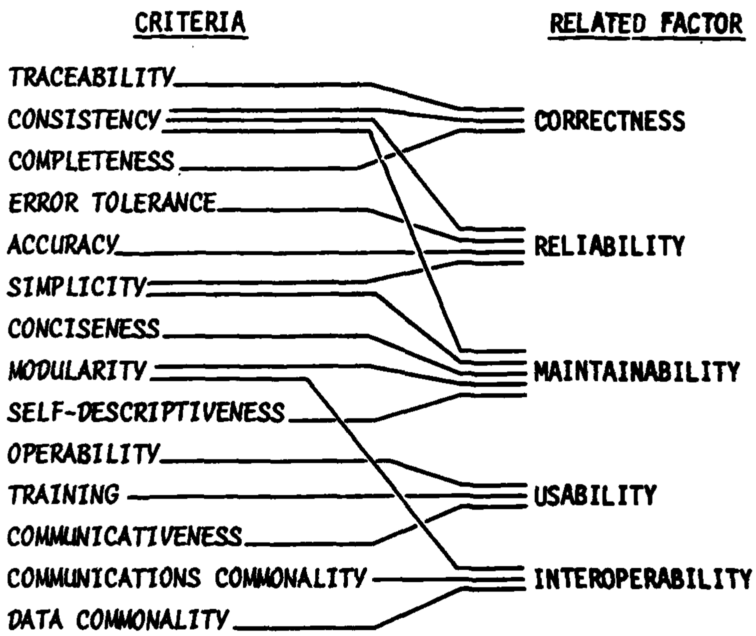
CRITERION	DEFINITION
EXECUTION EFFICIENCY	Those attributes of the software that provide for minimum processing time.
STORAGE EFFICIENCY	Those attributes of the software that provide for minimum storage requirements during operation.
ACCESS CONTROL	Those attributes of the software that provide for control of the access of software and data.
ACCESS AUDIT	Those attributes of the software that provide for an audit of the access of software and data.
OPERABILITY	Those attributes of the software that determine operation and procedures concerned with the operation of the software.
TRAINING	Those attributes of the software that provide transition from current operation or initial familiarization.
COMMUNICATIVENESS	Those attributes of the software that provide useful inputs and outputs which can be assimilated.
SOFTWARE SYSTEM INDEPENDENCE	Those attributes of the software that determine its dependency on the software environment (operating systems, utilities, input/output routines, etc.)
MACHINE INDEPENDENCE	Those attributes of the software that determine its dependency on the hardware system.
COMMUNICATIONS COMMONALITY	Those attributes of the software that provide the use of standard protocols and interface routines.
DATA COMMONALITY	Those attributes of the software that provide the use of standard data representations.
CONCISENESS	Those attributes of the software that provide for implementation of a function with a minimum amount of code.

2.3.2 EXAMPLE OF IDENTIFYING SOFTWARE CRITERIA

Continuing with the example of paragraph 2.2.2, the software criteria for the identified quality factors would be chosen.

2a. *Identify Critical Software Attributes*

Using the relationships provided in Table 2.3-1, the following criteria would be identified.



2b. *Provide Definitions*

The definitions for each of these criteria would be provided also.

2.4 PROCEDURES FOR ESTABLISHING QUANTIFIABLE GOALS

2.4.1 PROCEDURES

The last level, which is the most detailed and quantified, requires precise statements of the level of quality that will be acceptable for the software product.

Currently, the underlying mathematical relationships which allow measurement at this level of precision do not exist for all of the quality factors. The mechanism for making the precise statement for any quality factor is a rating of the factor. The underlying basis for the ratings is the effort or cost required to perform a function such as to correct or modify the design or program. For example, rating for maintainability might be that the average time to fix a problem should be five man-days or that 90% of the problem fixes should take less than six man-days. This rating would be specified as a quality requirement. To comply with this specification, the software would have to exhibit characteristics which, when present, given an indication that the software will perform to this rating. These characteristics are measured by metrics which are inserted into a mathematical relationship to obtain the predicted rating.

In order to choose ratings such as the two mentioned above, data must be available which allows the decision maker to know what is a "good rating" or perhaps what is the industry average. Currently there is generally a lack of good historical data to establish these expected levels of operations and maintenance performance for software. There are significant efforts underway to compile historical data and derive the associated performance statistics [DUVA76]. Individual software development organizations and System Program Offices should attempt to compile historical data for their particular environment. Any environment-unique data available should be used as a check against the data provided as guidelines in this manual. The data utilized in this section is based on experiences applying the metrics to several large command and control software systems and other experiences reported in the literature.

3a. *Specify Rating for Each Quality Factor*

After identification of the critical quality factors, specific performance levels or ratings required for each factor should be specified. Table 2.4.1 should be used as a guideline for identifying the ratings for the particular factors. Note that mathematical relationships have not been established for some of the factors. In those cases, it is advisable not to levy requirements for meeting a specific quality rating but instead specify the relative importance of the quality factor as a development goal. Not that the reliability ratings are provided in terms familiar to traditional hardware reliability. Just as in hardware reliability there are significant differences between ratings of .9 and .99.

3b. *Identify Specific Metrics to be Applied*

The next step or an alternative to 3a is to identify the specific metrics which will be applied to the various software products produced during the development. The Metric Worksheets described in Section 3 can be used for this purpose or Table 2.4-2 can be used to identify the metrics and reference can be made to RADC-TR-79- [MCCA79] where definitions of the metrics are provided.

3c. *Specification of Metric Threshold Values*

In lieu of specifying quality ratings or in addition to the ratings, specific minimum values for particular metrics may be specified. This technique is equivalent to establishing a standard which is to be adhered to. Violations to the value established are to be reported. Typical values can be derived by applying the metrics to software products developed in a particular environment or by looking at the scores reported in [MCCA77] and [MCCA79]. When establishing these threshold values based on past project data, projects which have been considered sucessful, i.e., have demonstrated good characteristics during their life cycle should be chosen. For example, a system which has been relatively cost-effective to maintain over its operational history should be chosen to apply the metrics related to maintainability and establish threshold values.

Table 2.4-1 Quality Factor Ratings

QUALITY FACTOR	RATING EXPLANATION	RATING GUIDELINES				
RELIABILITY *	Rating is in terms of the number of errors that occur after the start of formal testing. Rating = 1- Number of Errors / Number of Lines of source code excluding comments	RATING	.9	.98**	.99	.999
		ERRORS / 100 LOC	10	2	1	.1
MAINTAINABILITY *	Rating is in terms of the average amount of effort required to locate and fix an error in an operational program. Rating = 1-.1 (Average number of man days per fix)	RATING	.3	.5	.7**	.9
		AVERAGE EFFORT (MAN DAYS)	7	5	3	1
PORTABILITY *	Rating is in terms of the effort required to convert a program to run in another environment with respect to the effort required to orginally implement the program. Rating = 1- Effort to Transport / Effort to Implement	RATING	.25	.5**	.75	.9
		% OF ORIGINAL EFFORT	75	50	25	10
FLEXIBILITY *	Rating is in terms of the average effort required to extend a program to include other requirements. Rating = 1-.05 (Average number of man days to change)	RATING	.3	.5 **	.7	.9
		AVERAGE EFFORT (MAN DAYS)	14	10	6	2

(Continued)

Table 2.4-1 (Continued)

QUALITY FACTOR	RATING EXPLANATION
CORRECTNESS	The function which the software is to perform is incorrect. The rating is in terms of effort required to implement the correct function.
EFFICIENCY	The software does not meet performance (speed, storage) requirements. The rating is in terms of effort required to modify software to meet performance requirements.
INTEGRITY	The software does not provide required security. The rating is in terms of effort required to implement proper levels of security.
USABILITY	There is a problem related to operation of the software, the user interface, or the input/output. The rating is in terms of effort required to improve human factors to acceptable level.
TESTABILITY	The rating is in terms of effort requried to test changes or fixes.
REUSABILITY	The rating is in terms of effort required to use software in a different application.
INTEROPERABILITY	The rating is in terms of effort required to couple the system to another system.

NOTES

* Data collected to date provides some basis upon which to allow quantitative ratings for these quality factors. These ratings should be modified based on data collected within a specific development environment. Data has not been collected to support ratings of the other quality factors.

** Indicates rating which might be considered current industry average.

Table 2.4-2 Quality Metrics Related to Factors

FACTOR	METRICS *
CORRECTNESS	TRACEABILITY CHECK (TR.1) COMPLETENESS CHECKLIST (CP.1) CONSISTENCY CHECKLISTS (CS.1-2)
RELIABILITY	ACCURACY CHECKLIST (AY.1) ERROR TOLERANCE CHECKLISTS (ET.1-S) DESIGN STRUCTURE MEASURE (ST.1) STRUCTURED PROGRAMMING CHECK (SI.2) COMPLEXITY MEASURE (SI.3) CODING SIMPLICITY MEASURE (SI.4)
EFFICIENCY	PERFORMANCE REQUIREMENTS CHECK (EE.1) ITERATIVE PROCESSING MEASURE (EE.2) DATA USAGE MEASURE (EE.3) STORAGE EFFICIENCY MEASURE (EE.4)
INTEGRITY	ACCESS CONTROL CHECKLIST (AC.1) ACCESS AUDIT CHECKLIST (AA.1)
USABILITY	OPERABILITY CHECKLIST (OP.1) TRAINING CHECKLIST (TR.1) USER INPUT INTERFACE MEASURE (CM.1) USER OUTPUT INTERFACE MEASURE (CM.2)
MAINTAINABILITY	CONSISTENCY CHECKLISTS (CS1-2) DESIGN STRUCTURE MEASURE (SI.1) STRUCTURE LANGUAGE CHECK (SI.2) COMPLEXITY MEASURE (SI.3 CODING SIMPLICITY MEASURE (SI.4) STABILITY MEASURE (MO.1) MODULAR INPLEMENTATION MEASURE (MO.2) QUANTITY OF COMMENTS (SD.1) EFFECTIVENESS OF COMMENTS MEASURE (SD.2) DESCRIPTIVENESS OF IMPLEMENTATION LANGUAGE MEASURE (SD.3) CONCISENESS MEASURE (CO.1)

* Acronym references in parentheses relate to definitions in [MCCA79].

(Continued)

Table 2.4-2 Quality Metrics Related to Factors (Continued)

FACTOR	METRICS *
FLEXIBILITY	MODULAR IMPLEMENTATION MEASURE (MO.2) INTERFACE MEASURE (GE.1) GENERALITY CHECKLIST (GE.2) DATA STORAGE EXPANSION MEASURE (EX.1) EXTENSIBILITY MEASURE (EX.1) QUANTITY OF COMMENTS (SD.1) EFFECTIVENESS OF COMMENTS MEASURE (SD.2) DESCRIPTIVENESS OF IMPLEMENTATION LANGUAGE MEASURE (SD.3)
TESTABILITY	DESIGN STRUCTURE MEASURE (SI.1) STRUCTURED PROGRAMMING CHECK (SI.2) COMPLEXITY MEASURE (SI.3) CODING SIMPLICITY MEASURE (SI.4) STABILITY MEASURE (MO.1) MODULAR IMPLEMENTATION MEASURE (MO.2) TEST CHECKLISTS (IN.1-3) QUANTITY OF COMMENTS (SD.1) EFFECTIVENESS OF COMMENTS MEASURE (SD.2) DESCRIPTIVENESS OF IMPLEMENTATION LANGUAGE MEASURE (SD.3)
PORTABILITY	MODULAR IMPLEMENTATION MEASURE (MO.2) QUANTITY OF COMMENTS (SD.1) EFFECTIVENESS OF COMMENTS MEASURE (SD.2) DESCRIPTIVENESS OF IMPLEMENTATION LANGUAGE MEASURE (SD.3) SOFTWARE SYSTEM INDEPENDENCE MEASURE (SS.1) MACHINE INDEPENDENCE MEASURE (MI.1)
REUSEABILITY	MODULAR IMPLEMENTATION MEASURE (MO.2) INTERFACE MEASURE (GE.1) GENERALITY CHECKLIST (GE.2) QUANTITY OF COMMENTS (SD.1) EFFECTIVENESS OF COMMENTS MEASURE (SD.2) DESCRIPTIVENESS OF IMPLEMENTATION LANGUAGE MEASURE (SD.3) SOFTWARE SYSTEM INDEPENDENCE MEASURE (SS.1) MACHINE INDEPENDENCE MEASURE (MI.1)
INTEROPERABILITY	MODULAR IMPLEMENTATION MEASURE (MO.2) COMMUNICATIONS COMMONALITY CHECKLIST (CC.1) DATA COMMONALITY CHECKLIST (DC.1)

2.4.2 EXAMPLE OF METRICS

Using the example of paragraph 2.2.2, the quality ratings would be specified as follows.

3a *Specific Quality Factor Ratings*

Ratings for two of the five important quality factors can be established using Table 2.4-1.

Reliability	.99	Require less than one error per 100 lines of code to be detected during formal testing.
Maintainability	.8	Require 2 man days as an average level of maintenance for correcting an error.

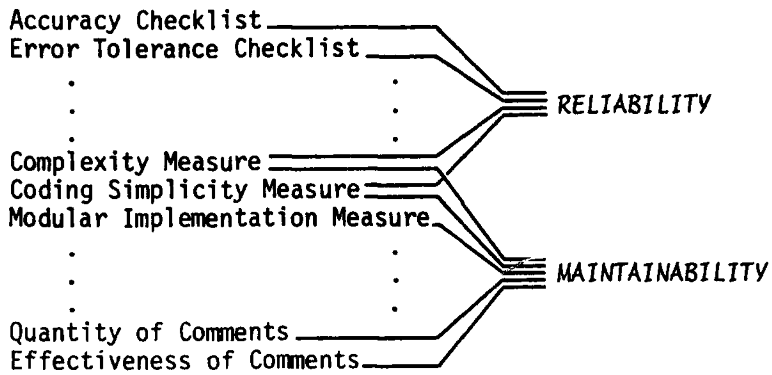
These ratings can also be established at each measurement period during the development as follows:

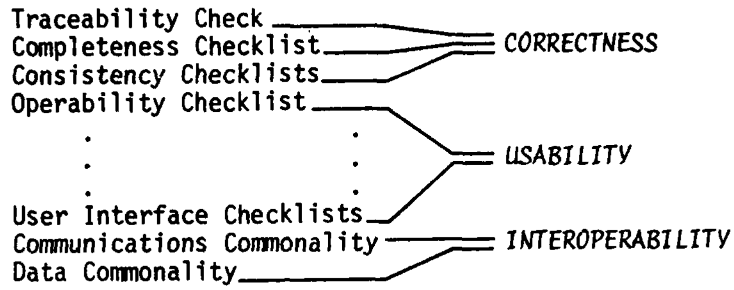
	REQ	POR	CDR	IMPL	ACCEPTANCE
Reliability	.8	.8	.9	.9	.99
Maintainability	.7	.7	.8	.8	.8

The progressively better scores are required because there is more detailed information in the later phases of the development to which to apply the metrics and more confidence in the metrics' indication of quality. This is analagous to the concept of reliability growth. For other quality factors see step 3b.

3b *Identify Specific Metrics to be Applied*

The metrics to be applied to assess the level of each important quality factor are chosen from Table 2.4-2. A subset is shown on the following page:





3c *Specify Threshold Values*

The following threshold values are established based on past experience and to provide a goal for the quality factors that were not given ratings. They were derived by determining the average scores of past applications of the metrics.

Quantity of Comments	.2
Effectiveness of Comments	.6
Complexity Measure	.1
Traceability Check	.9
Completeness Checklist	1.
Consistency Checklist	.9
Operability Checklist	.6
Training Checklist	.75
User Interface Checklist	.75
Communications Commonality	.8
Data Commonality	.8

2.5 EVALUATION OF DEVELOPMENT PLAN

In an acquisition environment the initial benefits of utilizing the quality metrics concepts are realized in the source selection process. The acquisition office should include the quality goals established as software requirements in the Request for Proposal. The software attributes should be also identified as required characteristics in the software and the metrics as the vehicles for assessing their existence. The bidders should be required to describe how they plan to provide those characteristics in the software. This discussion should be provided in the portion of the proposal that describes their development plan.

The description of the bidder's approach for including the required attributes in the software not only forces acknowledgement of these additional requirements but also provides additional information with which to evaluate the bidders during source selection.

SECTION 3

PROCEDURES FOR APPLYING MEASUREMENTS

3.1 WHEN TO APPLY MEASUREMENTS

The software quality metrics are oriented toward the availability of information about the software system as it progresses in its development. In the early phases of the development, the metrics are applied to the documentation produced to describe the concepts of the system and its design. In the later phases the metrics are oriented not only to documentation but also to the source code that is available.

Thus, the application of the metrics logically follows the phased development of software. The first application of the metric is at the end of the requirements analysis. The next application is during design. If the design phase has been decomposed into a preliminary design phase and a detailed design phase, the metrics should be applied at the end of each of those phases. During implementation, i.e. coding the metrics oriented toward the source code should be applied periodically to assess the quality growth exhibited as the code evolves. The timing of the application of the metrics is shown in Figure 3.1-1. The application of the metrics can be done during or just proir to formal customer reviews (as shown in Figure 3.1-1) or during equivalent activities conducted by the development personnel.

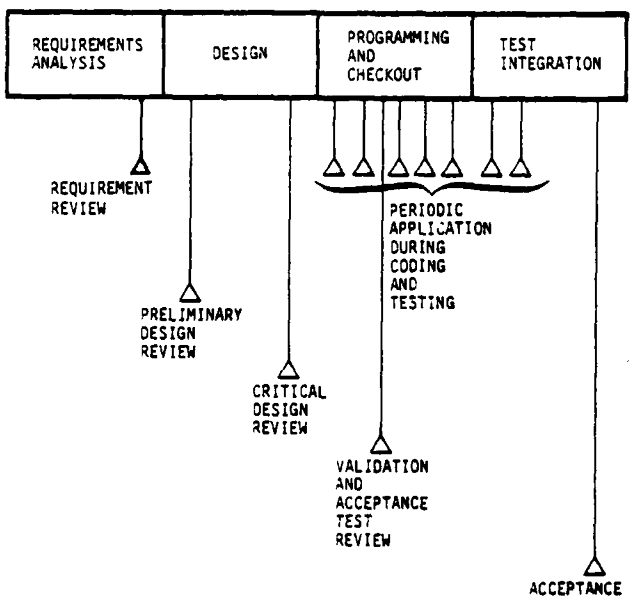


Figure 3.1-1 Timing of Metrics Application

3.2 SOURCES OF QUALITY INFORMATION

A typical minimum set of documents and source code are shown in Figure 3.2-1. These documents plus the source code are the sources of the quality information derived from the application of the metrics.

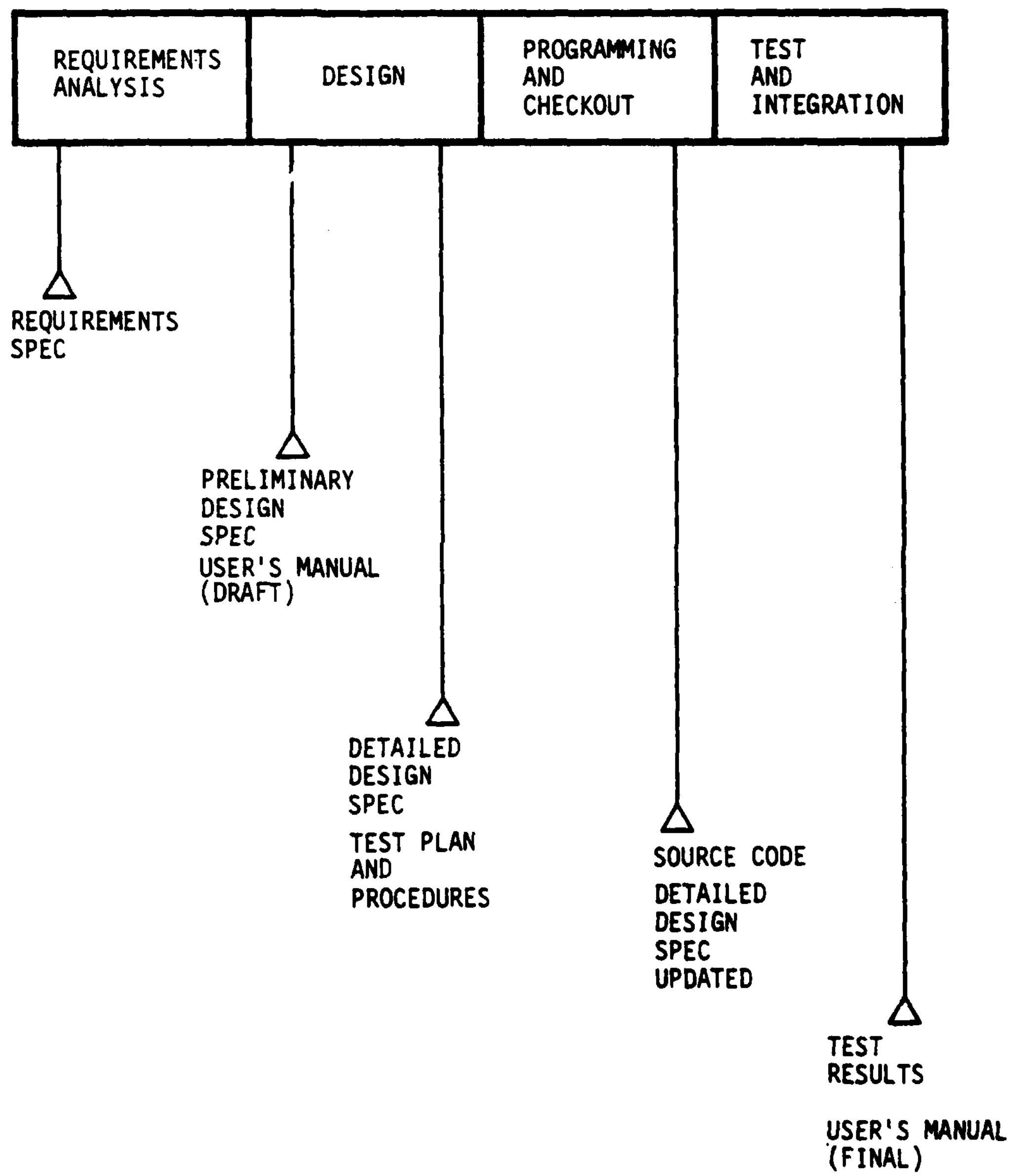


Figure 3.2-1

3.3 APPLICATION OF THE MEASUREMENTS

The vehicle for applying the software metrics are the metric worksheets contained in this section of the manual. The procedure is to take the available documentation/source code, apply the appropriate worksheet (as shown in Figure 3.3-1), and translate the measurements to metric scores.

The worksheets are organized as follows. In the header portion of the worksheet is the worksheet identification which (1) identifies the phase during which the worksheet is initially used and the level (system or module) to which the worksheet applies, (2) provides identification of the system and the module to which the worksheet has been applied, and (3) provides identification of the date and the inspector who took the measurements. The remaining portion of each worksheet contains the measurements to be taken and questions to be answered. These measurements and questions are organized by quality factors identified in parentheses. Each logical group of measurements and questions have a group identifier and are bordered. When applying the measurements, only those in the groups that relate to the quality factors chosen as quality goals should be applied.

Metric Worksheet # 1 and # 2a contain system level metrics and are applied at the system or major subsystem (CPCI) level to the System Requirements Specification, the Preliminary Design Specification, the User's Manual, and the Test documentation.

Metric Worksheets # 2b and #3 contain module level metrics and are applied to each module's design (Detailed Design Specification) and implementation (source code).

Definitions and interpretations of the individual measurements contained in the worksheets are found in Appendix B of the first volume. Next to each measurement element on the worksheets is an index into the metric table in Appendix B.

As shown in Figure 3.3-1, the worksheets may be applied several times during the development. For example, Metric Worksheet #2b which is applied for each module to the detailed design document during design is also applied to the detailed design document after it has been updated to reflect the actual implementation. The worksheet does not have to be totally reapplied. The successive applications of any worksheet should require considerably less effort than the original application. The successive applications should involve simply updates to reflect the updates made to the system since the previous application of the worksheet.

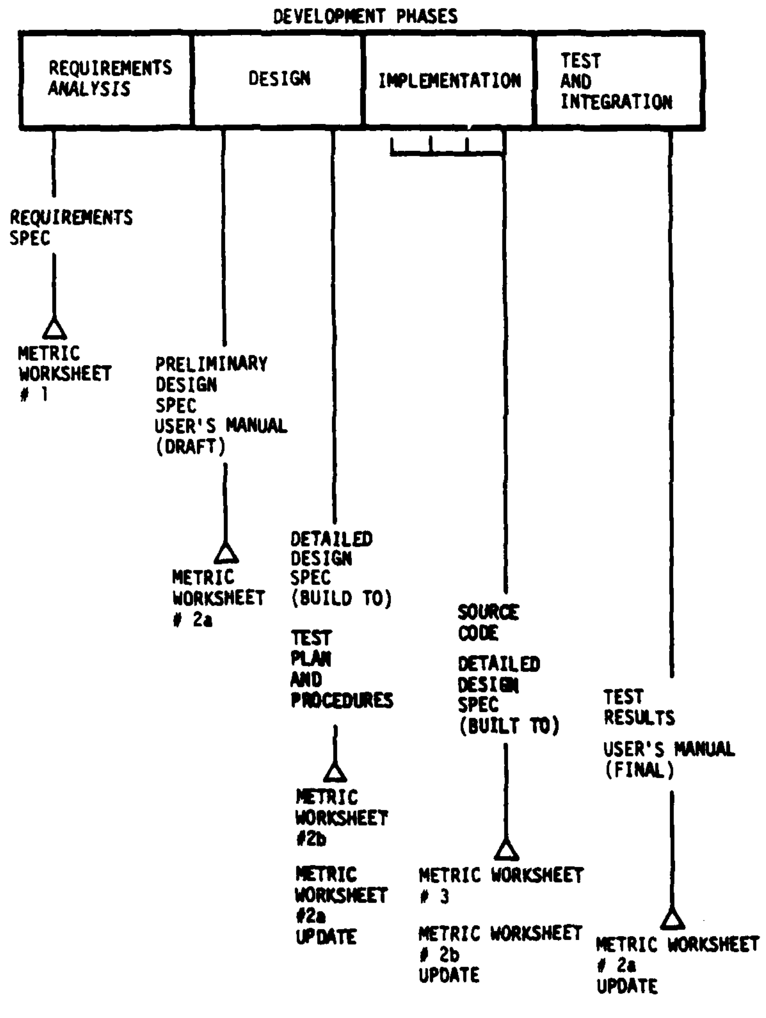


Figure 3.3-1 Application of the Metric Worksheets

METRIC WORKSHEET 1 REQUIREMENTS ANALYSIS/SYSTEM LEVEL	SYSTEM NAME:________	DATE ________ INSPECTOR:________

I. COMPLETENESS (CORRECTNESS, RELIABILITY)

1.	Number of major functions identified (equivalent to CPCI). CP.1		
2.	Are requirements itemized so that the various functions to be performed, their inputs and outputs, are clearly delineated? CP.1(1)	Y	N
3.	Number of major data references. CP.1(2)		
4.	How many of these data references are not defined? CP.1(2)		
5.	How many defined functions are not used? CP.1(3)		
6.	How many referenced functions are not defined? CP.1(4)		
7.	How many data references are not used? CP.1(2)		
8.	How many referenced data references are not defined? CP.1(6)		
9.	Is the flow of processing and all decision points in that flow described? CP.1(5)	Y	N
10.	How many problem reports related to the requirements have been recorded? CP.1(7)		
11.	How many of those problem reports have been closed (resolved)? CP.1(7)		

II. PRECISION (RELIABILITY)

1.	Has an error analysis been performed and budgeted to functions? AY.1(1)	Y	N
2.	Are there definitive statements of the accuracy requirements for inputs, outputs, processing, and constants? AY.1(2)	Y	N
3.	Are there definitive statements of the error tolerance of input data? ET.2(1)	Y	N
4.	Are there definitive statements of the requirements for recovery from computational failures? ET.3(1)	Y	N
5.	Is there a definitive statement of the requirement for recovery from hardware faults? ET.4(1)	Y	N
6.	Is there a definitive statement of the requirements for recovery from device errors? ET.5(1)	Y	N

III. SECURITY (INTEGRITY)

1.	Is there a definitive statement of the requirements for user input/output access controls? AC.1(1)	Y	N
2.	Is there a definitive statement of the requirements for data base access controls? AC.1(2)	Y	N
3.	Is there a definitive statement of the requirements for memory protection across tasks? AC.1(3)	Y	N
4.	Is there a definitive statement of the requirements for recording and reporting access to system? AA.1(1)	Y	N
5.	Is there a definitive statement of the requirements for immediate indication of access violation? AA.1(2)	Y	N

METRIC WORKSHEET 1 REQUIREMENTS ANALYSIS/SYSTEM LEVEL	SYSTEM NAME: ______	DATE ______ INSPECTOR: ______

IV. HUMAN INTERFACE (USABILITY)

1. Are all steps in the operation described (operations concept)? OP.1(1)	Y	N
2. Are all error conditions to be reported to operator/user identified and the responses described? OP.1(2)	Y	N
3. Is there a statement of the requirement for the capability to interrupt operation, obtain status, modify, and continue processing? OP.1(3)	Y	N
4. Is there a definitive statement of requirements for optional input media? CM.1(6)	Y	N
5. Is there a definitive statement of requirements for optional output media? CM.2(7)	Y	N
6. Is there a definitive statement of requirements for selective output control? CM.2(1)	Y	N

V. PERFORMANCE (EFFICIENCY)

1. Have performance requirements (storage and run time) been identified for the functions to be performed? EE.1	Y	N

VI. SYSTEM INTERFACES (INTEROPERABILITY)

1. Is there a definitive statement of the requirements for communication with other systems? CC.1(1)	Y	N
2. Is there a definitive statement of the requirements for standard data representations for communication with other systems? DC.1(1)	Y	N

VII. INSPECTOR'S COMMENTS

Make any general or specific comments that relate to the quality observed while applying this checklist.

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? TR.1	Y N
2.	How many major functions (CPCIS) are identified? CP.1	
3.	How many functions identified are not defined? CP.1(2)	
4.	How many defined functions are not used? CP.1(3)	
5.	How many interfaces between functions are not defined? CP.1(6)	
6.	Number of total problem reports recorded? CP.1(7)	
7.	Number of those reports that have not been closed (resolved?) CP.1(7)	
8.	Profile of problem reports: (number of following types)	

a.	Computational	
b.	Logic	
c.	Input/output	
d.	Data handling	
e.	OS/System Support	
f.	Configuration	
g.	Routine/Routine interface	
h.	Routine/System Interface	
i.	Tape Processing	
j.	User interface	
k.	data base interface	
l.	user requested changes	
m.	Preset data	
n.	Global variable definition	
p.	Recurrent errors	
q.	Documentation	
r.	Requirement compliance	
s.	Operator	
t.	Questions	
u.	Hardware	

II. PRECISION (RELIABILITY)

1.	Have math library routines to be used been checked for sufficiency with regards to accuracy requirements? AY.1(3)	Y N
2.	Is concurrent processing centrally controlled? ET.1(1)	Y N
3.	How many error conditions are reported by the system? ET.1(2)	Y N
4.	How many of those errors are automatically fixed or bypassed and processing continues? ET.1(2)	
5.	How many, require operator intervention? ET.1(2)	
6.	Are provisions for recovery from hardware faults provided? ET.4(2)	Y N
7.	Are provisions for recovery from device errors provided? ET.5(2)	Y N

III STRUCTURE (RELIABILITY, MAINTAINABILITY, TESABILITY, PORTABILITY, REUSABILITY, INTEROPERABILITY)

1.	Is a hierarchy of system, identifying all modules in the system provided? SI.1(1)	Y N
2.	Number of Modules SI.1(2)	
3.	Are there any duplicate functions? SI.1(2)	Y N
4.	Based on hierarchy or a call/called matrix, how many modules are called by more than one other module? GE.1	
5.	Are the constants used in the system defined once? GE.2(5)	Y N

METRIC WORKSHEET 2a DESIGN/SYSTEM LEVEL	SYSTEM NAME______	DATE:______ INSPECTOR:______

IV. OPTIMIZATION (EFFICIENCY)

1. Are storage requirements allocated to design? SE.1(1)	Y	N
2. Are virtual storage facilities used? SE.1(2)	Y	N
3. Is dynamic memory management used? SE.1(5)	Y	N
4. Is a performance optimizing compiler used? SE.1(7)	Y	N
5. Is global data defined once? CS.2(3)	Y	N
6. Have Data Base or files been organized for efficient processing? EE.3(5)	Y	N
7. Is data packing used? EE.2(5)	Y	N
8. Number of overlays EE.2(4)		
9. Overlay efficiency - memory allocation EE.2(4)		
max overlay size		
min overlay size		

V. SECURITY (INTEGRITY)

1. Are user Input/Output access controls provided? AC.1(1)	Y	N
2. Are Data Base access controls provided? AC.1(2)	Y	N
3. Is memory protection across tasks provided? AC.1(3)	Y	N
4. Are there provisions for recording and reporting errors? AC.2(1,2)	Y	N

VI. SYSTEM INTERFACES (INTEROPERABILITY)

1. How many other systems will this system interface with? CC.1(1)		
2. Have protoc l standards been established? CC.1(2)	Y	N
3. Are they being complied with? CC.1(2)	Y	N
4. Number of modules used for input and output to other systems? CC.1(3,4)	Y	N
5. Has a standard data representation been established or translation standards between representations been established? DC.1(1)	Y	N
6. Are they being complied with? DC.1(2)		
7. Number of modules used to perform translations? DC.1(3)		

VII. HUMAN INTERFACE (USABILITY)

1. Are all steps in operation described including alternative flows? OP.1(1)	Y	N
2. Number of operator actions? OP.1(4)		

METRIC WORKSHEET 2a DESIGN/SYSTEM LEVEL	SYSTEM NAME:	DATE: INSPECTOR

VII. HUMAN INTERFACE (USABILITY) Continued

3. Estimated or Actual time to perform? OP.1(4)		
4. Budgeted time for complete job? OP.1(4)		
5. Are job set up and tear down procedures described? OP.1(5)		
6. Is a hard copy of operator interactions to be maintained? OP.1(6)	Y	N
7. Number of operator messages and responses? OP.1(2)		
8. Number of different formats? OP.1(2)		
9. Are all error conditions and responses appropriately described? OP.1(2)		
10. Does the capability exist for the operator to interrupt, obtain status, save, modify, and continue processing? OP.1(3)	Y	N
11. Are lesson plans/training materials for operators, end users, and maintainers provided? TN.1(1)	Y	N
12. Are realistic, simulated exercises provided? TN.1(2)	Y	N
13. Are help and diagnostic information available? TN.1(3)		
14. Number of input formats CM.1(2)		
15. Number of input values CM.1(1)		
16. Number of default values CM.1(1)		
17. Number of self-identifying input values CM.1(3)		
18. Can input be verified by user prior to execution? CM.1(4)		
19. Is input terminated by explicitly defined by logical end of input? CM.1(5)	Y	N
20. Can input be specified from different media? CM.1(6)	Y	N
21. Are there selective output controls? CM.2(1)	Y	N
22. Do outputs have unique descriptive user oriented labels? CM.2(5)	Y	N
23. Do outputs have user oriented units? CM.2(3)	Y	N
24. Number of output formats? CM.2(4)		
25. Are logical groups of output separated for user emamination? CM.2(5)	Y	N
26. Are relationships between error messages and outputs unambiguous? CM.2(6)	Y	N
27. Are there provisions for directing output to different media? CM.2(7)	Y	N

VIII. TESTING (TESTABILITY) APPLY TO TEST PLAN, PROCEDURES, RESULTS

1. Number of paths? IN.1(1)		4. Number of input parameters to be tested? IN.1(2)	
2. Number of paths to be tested? IN.1(1)			
3. Number of input parameters? IN.1(1)		5. Number of interfaces? IN.2(1)	

METRIC WORKSHEET 2a DESIGN/SYSTEM LEVEL	SYSTEM NAME: ______________	DATE: INSPECTOR: __________

VIII. TESTING (TESTABILITY) - APPLY TO TEST PLAN, PROCEDURES, RESULTS (CONTINUED)

6. Number of interfaces to be tested? IN.2(1)		9. Number of modules? IN.3(1)	
7. Number of itemized performance requirements? IN.2(2)		10. Number of modules to be exercised? IN.3(1)	
8. Number of performance requirements to be tested? IN.2(2)		11. Are test inputs and outputs provided in summary form? IN.3(2)	

IX DATA BASE

1. Number of unique data items in data base SI.1(6)	
2. Number of preset data items SI.1(6)	
3. Number of major segments (files) in data base SI.1(7)	

X INSPECTOR'S COMMENTS

Make any general or specific comments about the quality observed while applying this checklist.

METRIC WORKSHEET 2b DESIGN/MODULE LEVEL	SYSTEM NAME: ______________ MODULE NAME: ______________	DATE: ____________ INSPECTOR: ________

I. COMPLETENESS (CORRECTNESS, RELIABILITY)

1. Can you clearly distinguish inputs, outputs, and the function being performed? CP.1(1) — Y N
2. How many data references are not defined, computed, or obtained from an external source? CP.1(2)
3. Are all conditions and processing defined for each decision point? CP.1(5) — Y N
4. How many problem reports have been recorded for this module? CP.1(7)
5. Profile of Problem Reports:
6. Number of problem reports still outstanding CP.1(7)

a. Computational
b. Logic
c. Input/Output
e. System/OS Support
f. Configuration
g. Routine/Routine Interface
h. Routine/System Interface
i. Tape Processing
j. User Interface
k. Data Base Interface
l. User Requested Changes
m. Preset Data
n. Global Variable Definition
p. Recurrent Errors
q. Documentation
r. Requirement Compliance
t. Operator
u. Questions
v. Hardware

II. PRECISION (RELIABILITY)

1. When an error condition is detected, is it passed to calling module? ET.1(3) — Y N
2. Have numerical techniques being used in algorithm been analyzed with regards to accuracy requirements? AY.1(4) — Y N
3. Are values of inputs range tested? ET.2(2) — Y N
4. Are conflicting requests and illegal combinations identified and checked? ET.2(3) — Y N
5. Is there a check to see if all necessary data is available before processing begins? ET.2(5) — Y N
6. Is all input checked, reporting all errors, before processing begins? ET.2(4) — Y N
7. Are loop and multiple transfer index parameters range tested before use? ET.3(2) — Y N
8. Are subscripts range tested before use? ET.3(3) — Y N
9. Are outputs checked for reasonableness before processing continues? ET.3(4) — Y N

III. STRUCTURE (RELIABILITY, MAINTAINABILITY, TESTABILITY)

1. How many Decision Points are there? SI.3
2. How many subdecision Points are there? SI.3
3. How many conditional branches are there? SI.3
4. How many unconditional branches are there? SI.3

METRIC WORKSHEET 2B DESIGN/MODULE LEVEL	SYSTEM NAME: ________ MODULE NAME: ________	Pg. 2

III. STRUCTURE (RELIABILITY, MAINTAINABILITY, TESTABILITY) (CONTINUED)

5. Is the module dependent on the source of the input or the destination of the output? SI.1(3)	Y	N
6. Is the module dependent on knowledge of prior processing? SI.1(3)	Y	N
7. Are any limitations of the processing performed by the module identified? EX.2(1)	Y	N
8. Number of entrances into modules SI.1(5)		
9. Number of exits from module SI.1(5)		

IV. REFERENCES (MAINTAINABILITY, FLEXIBILITY, TESTABILITY, PORTABILITY, REUSABILITY, INTEROPERABILITY)

1. Number of references to system library routines, utilities or other system provided facilities SS.1(1)		
2. Number of input/output actions MI.1(2)		
3. Number of calling sequence parameters MO.2(3)		
4. How many calling sequence parameters are control variables MO.2(3)		
5. Is input passed as calling sequence parameters MO.2(4)	Y	N
6. Is output passed back to calling module? MO.2(5)	Y	N
7. Is control returned to calling module MO.2(6)	Y	N
8. Is temporary storage shared with other modules? MO.2(7)		
9. Does the module mix input, output and processing functions in same module? GE.2(1)		
10. Number of machine dependent functions performed? GE.2(2)		
11. Is processing data volume limited? GE.2(3)	Y	N
12. Is processing data value limited? GE.2(4)	Y	N
13. Is a common, standard subset of programming language to be used? SS.1(2)	Y	N
14. Is the programming language available in other machines? MI.1(1)	Y	N

V. EXPANDABILITY (FLEXIBILITY)

1. Is logical processing independent of storage specification? EX.1(1)	Y	N
2. Are accuracy, convergence, or timing attributes parametric? EX.2(1)	Y	N
3. Is module table driven? EX.2(2)	Y	N

VI. OPTIMIZATION (EFFICIENCY)

1. Are specific performance requirements (storage and routine) allocated to this module? EE.1	Y	N

METRIC WORKSHEET 2b DESIGN/MODULE LEVEL	SYSTEM NAME: ____________ MODULE NAME: ____________	Pg. 3

VI. OPTIMIZATION (EFFICIENCY) (CONTINUED)

2. Which category does processing fall in: EE.2

   ☐ Real-time
   ☐ On-line
   ☐ Time-constrained
   ☐ Non-time critical

3. Are non-loop dependent functions kept out of loops? EE.2(1) ☐
4. Is bit/byte packing/unpacking performed in loops? EE.2(5) ☐
5. Is data indexed or reference efficiently? EE.3(5) ☐

VII. FUNCTIONAL CATEGORIZATION

Categorize function performed by this module according to following:

☐ CONTROL - an executive module whose prime function is to invoke other modules.

☐ INPUT/OUTPUT - a module whose prime function is to communicate data between the computer and the user.

☐ PRE/POSTPROCESSOR - a module whose prime function is to prepare data for or after the invocation of a computation or data management module.

☐ ALGORITHM - a module whose prime function is computation.

☐ DATA MANAGEMENT - a moudle whose prime function is to control the flow of data within the computer.

☐ SYSTEM - a module whose function is the scheduling of system resources for other modules.

VIII. CONSISTENCY

1.	Does the design representation comply with established standards CS.1(1)	Y	N
2.	Do input/output references comply with established standards CS.1(3)	Y	N
3.	Do calling sequences comply with established standards CS.1(2)	Y	N
4.	Is error handling done according to established standards CS.1(4)	Y	N
5.	Are variable named according to established standards CS.2(2)	Y	N
6.	Are global variables used as defined globally CS.2(3)	Y	N

METRIC WORKSHEET 2b DESIGN/MODULE LEVEL	SYSTEM NAME: __________ MODULE NAME: __________	Pg. 4

IX. INSPECTOR'S COMMENTS

Make any specific or general comments about the quality observed while applying this checklist?

METRIC WORKSHEET 3 SOURCE CODE/MODULE LEVEL	SYSTEM NAME: ______ MODULE NAME: ______	DATE: ______ INSPECTOR: ______

I. STRUCTURE (RELIABILITY, MAINTAINABILITY, TESTABILITY)

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

II. CONCISENESS (MAINTAINABILITY) - SEE SUPPLEMENT

1. Number of operators CO.1 [ ]
2. Number of unique operators CO.1 [ ]
3. Number of Operands CO.1 [ ]
4. Number of unique operands CO.1 [ ]

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

1. Number of lines of comments SD.1 [ ]
2. Number of non-blank lines of comments SD.1 [ ]
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? SD.2(1) [Y | N]
5. How many decision points and transfers of control are not commented? SD.2(3) [ ]
6. Is all machine language code commented? SD.2(4) [Y | N]
7. Are non-standard HOL statements commented? SD.2(5) [Y | N]
8. How many declared variables are not described by comments? SD.2(6) [ ]
9. Are variable names (mnemonics) descriptive of the physical or functional property they represent? SD.3(2) [Y | N]
10. Do the comments do more than repeat the operation? SD.2(7) [Y | N]
11. Is the code logically blocked and indented? SD.3(3) [Y | N]
12. Number of lines with more than 1 statement. SD.3(4) [ ]
13. Number of continuation lines SD.3(4) [ ]

METRIC WORKSHEET 3 SOURCE CODE/MODULE LEVEL	SYSTEM NAME: ______ MODULE NAME: ______	Pg. 2

IV. INPUT/OUTPUT (RELIABILITY, FLEXIBILITY, PORTABILITY)

1. Number of input statements MI.1(2) [ ]
2. Number of output statements MI.1(2) [ ]
3. Is amount of input that can be handled parametric? GE.2(3) [Y] [N]
4. Are inputs range-tested (for inputs via calling sequences, global data, and input statements) ET.2(2) [Y] [N]
5. Are possible conflicts or illegal combinations in inputs checked? ET.2(3) [Y] [N]
6. Is there a check to determine if all data is available prior to processing? ET.2(5) [Y] [N]

V. REFERENCES (RELIABILITY, MAINTAINABILITY, TESTABILITY, FLEXIBILITY, PORTABILITY, REUSABILITY)

1. Number of calls to other modules MO.2(1) [ ]
2. Number of references to system library routines, utilities, or other system provided functions SS.1(1) [ ]
3. Number of calling sequence parameters MO.2(3) [ ]
4. How many elements in calling sequences are not parameters? MO.2(3) [ ]
5. How many of the calling parameters (input) are control variables? MO.2(3) [ ]
6. How many parameters passed to or from other modules are not defined in this module? MO.2(3) [ ]
7. Is input data passed as parameter? MO.2(4) [Y] [N]
8. Is output data passed back to calling module? MO.2(5) [Y] [N]
9. Is control returned to calling module? MO.2(6) [Y] [N]

VI. DATA (CORRECTNESS, RELIABILITY, MAINTAINABILITY, TESTABILITY)

1. Number of local variables SI.4(10) [ ]
2. Number of global variables SI.4(10) [ ]
3. Number of global variables renamed SE.1(3) [ ]
4. How many global variables are not used consistently with respect to units or type? CS.2(4) [ ]
5. How many variables are used for more than one purpose? CS.2(3) [ ]

VII. ERROR HANDLING - (RELIABILITY)

1. How many loop and multiple transfer index parameters are not range tested before use? ET.3(2) [ ]
2. Are subscript values range tested before use? ET.3(3) [Y] [N]
3. When an error condition occurs, is it passed to the calling module? ET.1(3) [Y] [N]
4. Are the results of a computation checked before outputting or before processing continues? ET.3(4) [Y] [N]

VIII. (EFFICIENCY)

1. Number of mix mode expressions? EE.3(3) [ ]
2. How many variables are initialized when declared? EE.3(2) [ ]
3. How many loops have non-loop dependent statements in them? EE.2(1) [ ]
4. How many loops have bit/byte packing/unpacking? EE.2(5) SE.1(6) [ ]
5. How many compound expressions defined more than once? EE.2(3) [ ]

METRIC WORKSHEET 3 SOURCE CODE/MODULE LEVEL	SYSTEM NAME: ____________ MODULE NAME: ____________	Pg. 3

IX. PORTABILITY

1. Is code independent of word and character size? MI.1(3) — Y / N
2. Number of lines of machine language statements. MI.1 — ______
3. Is data representation machine independent? MI.1(4) — Y / N
4. Is data access/storage system software independent? SS.1 — Y / N

X. FLEXIBILITY

1. Is module table driven EX.2(2) — Y / N
2. Are there any limits to data values that can be processed? GE.2(4) — Y / N
3. Are there any limits to amounts of data that can be processed? GE.2(3) — Y / N
4. Are accuracy, convergence and timing attributes parametric? EX.2(1) — Y / N

XI. DYNAMIC MEASUREMENTS (EFFICIENCY, RELIABILITY)

1. During execution are outputs within accuracy tolerances? AY.1(5) — Y / N
2. During module/development testing, what was run time? EX.2(3) — ______
3. Complete memory map for execution of this module SE.1(4)

	Size (words of memory)
APPLICATION	
SYSTEM	
DATA	
OTHER	

4. During execution how many data items were referenced but not modified EE.3(6) — ______
5. During execution how many data items were modified EE.3(7) — ______

XII. INSPECTORS COMMENTS

Make any general or specific comments that relate to the quality observed by you while applying this checklist:

3.4 TECHNIQUES FOR APPLYING MEASUREMENTS

Section 1.5 identified organizational approaches for utilizing the quality metric concepts during a software development. These approaches included both acquisition environments and internal development environments. The purpose of this section is to describe, at a lower level, how the metrics would be applied in either case.

The first technique for applying the metrics is by formal inspection. The formal inspection is performed by personnel of an organization independent of the development organization (the acquisition office, an independent quality assurance group, or an independent contractor). The metric worksheets are applied to delivered products at scheduled times and the results are formally reported.

The second technique is to utilize the worksheets during structured design and code walkthroughs held by the development team. A specific participant of the walkthrough can be designated for applying the worksheets and reporting any deficiencies during the walkthrough or a representative of the quality assurance organization can participate in the walkthrough with the purpose of taking the measurements of the design or code.

The last technique is for the development team to utilize the worksheets as guidelines, self-evaluations or in a peer review mode to enhance the qualtiy of the products they produce.

SECTION 4

PROCEDURES FOR ASSESSING THE QUALITY OF THE SOFTWARE PRODUCT

4.1 INTRODUCTION

The benefits of applying the software quality metrics are realized when the information gained from their application is analyzed. The analyses that can be done based on the metric data are described in the subsequent paragraphs. There are three levels at which analyses can be performed. These levels are related to the level of detail to which the quality assurance organization wishes to go in order to arrive at a quality assessment.

4.2 INSPECTOR'S ASSESSMENT

The first level at which an assessment can be made relies on the discipline and consistency introduced by the application of the worksheets. An inspector, using the worksheets, asks the same questions and takes the same counts for each module's source code or design document,etc. that is reviewed. Based on this consistent evaluation, a subjective comparison of products can be made.

1a. *Document Inspector's Assessment*

The last section in each worksheet is a space for the inspector to make comments on the quality observed while applying the worksheet. Comments should indicate an overall assessment as well as point out particular problem areas such as lack of comments, inefficiencies in implementation, or overly complex control flow.

1b. *Compile Assessments for System Review*

By compiling all of the inspector's assessments on the various documents and source code available at any time during the development, deficiencies can be identified.

4.3 SENSITIVITY ANALYSIS

The second level of detail utilizes experience gained through the application of metrics and the accumulation of historical information to take advantage of the quantitative nature of the metrics. The values of the measurements are used as indicators for evaluation of the progress toward a high quality product.

At appropriate times during a large-scale development, the application of the worksheets allows calculation of the metrics. The correspondence of the worksheets to the metrics is shown in Appendix B of [MCCA79]. The results of these calculations is a matrix of measurements. The metrics that have been established to date are at two levels-system level and module level. The approach to be described is applicable to both levels and will be described in relationship to the module level metric.

A n by k matrix of measurements results from the application of the metrics to the existing products of the development (e.g., at design, the products might include review material, design specifications, test plans, etc.) where there are k modules and n module level measurements applicable at this particular time.

$$M_d^m = \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1k} \\ m21 & & \ddots & \\ \vdots & & & \\ m_{n1} & & & m_{nk} \end{bmatrix}$$

This matrix represents a profile of all of the modules in the system with respect to a number of characteristics measured by the metrics. The analyses that can be performed are described in the following steps:

2a. *Assess Variation of Measurements*

Each row in the above matrix represents how each module in the system scored with respect to a particular metric. By summing all the values and calculating the average and standard deviation for that metric, each individual module's score can then be compared with the average and standard deviation. Those modules that score less than one standard deviation from the average should be identified for further examination. These calculations are illustrated below:

for metric i; Average Score = $A_i = \sum_{j=1}^{k} M_{ij}/k$

Standard Deviation = $\sigma_i = \sum_{j=1}^{k} (M_{ij}-A_i)^2/k$

Report Module j if $M_{ij} < A_i - \sigma_i$

2b. *Assess Low System Scores*

In examining a particular measure across all modules, consistently low scores may exist. It may be that a design or implementation technique used widely by the development team was the cause. This situation identifies the need for a new standard or stricter enforcement of existing standards to improve the overall development effort.

2c. *Assess Scores Against Thresholds*

As experience is gained with the metrics and data is accumulated, threshold values or industry acceptable limits may be established. The scores, for each module for a particular metric should be compared with the established threshold. A simple example is the percent of comments per line of source code. Certainly code which exhibits only one or two percent measurements for this metric would be identified for corrective action. It may be that ten percent is a minimum acceptable level. Another example is the complexity measure. A specific value of the complexity measure greater than some chosen value should be identified for corrective action

Report Module j if $M_{ij} < T_i$ (or $>T$ for complexity measures)

Where T_i = threshold value specified for metric i.

4.4 USE OF NORMALIZATION FUNCTION TO ASSESS QUALITY

The last level of assessing Quality is using the normalization functions to predict the quality in quantitative terms. The normalization functions are utilized in the following manner.

For a particular time there is an associated matrix of coefficients which represent the results of linear multivariate regression analyses against empirical data (past software developments). These coefficients, when multiplied by the measurement matrix results in an evaluation (prediction) of the quality of the product based on the development to date. This coefficient matrix, shown below, has n columns for the coefficients of the various metrics and 11 rows for the 11 quality factors.

$$C_d^m = \begin{bmatrix} c_{11} & c_{12} & . & . & . & c_{1n} \\ . & . & & & & \\ . & & . & & & \\ . & & & . & & \\ c_{11,1} & & & & & c_{11,n} \end{bmatrix}$$

To evaluate the current degree or level of a particular quality factor, i, for a module, j, the particular column in the measurement matrix is multiplied by the row in the coefficient matrix. The resultant value:

$$c_{i,1}\ m_{i,j} + c_{i,2}\ m_{2,j}\ .\ .\ . + c_{i,n}\ m_{m,j} = r_{i,j}$$

is the current predicted rating of that module, j for the quality factor, i. This predicted rating is then compared to the previously established rating to determine if the quality is as least as sufficient as required. The coefficient matrix should be relatively sparse (many $C_{ij} = 0$). Only subsets of the entire set of metrics applicable at any one time relate to the criteria of any particular quality factor.

Multiplying the complete measurement matrix by the coefficient matrix results in a ratings matrix. This matrix contains the current predicted ratings of each module for each quality factor. Each module then can be compared with the preset rating for each quality factor.

$$CM = R_d^m = \begin{bmatrix} r_{11} & r_{12} & . & . & . & r_{1,k} \\ . & . & & & & \\ . & & . & & & \\ r_{11,1} & & & & . & r_{11,k} \end{bmatrix}$$

This approach represents the most formal approach to evaluating the quality of a product utilizing the software quality metrics. Because the coefficient matrix has been developed only for a limited sample in a particular environment, it is neither generally applicable nor has statistical confidence in its value been achieved.

To use the normalization functions that currently exist the following steps should be performed.

3a. *Apply Normalization Function*

Table 4.4-1 contains the normalization functions that currently exist. If any of the quality factors identified in that table have been specified as a requirement of the development, then the metrics identified in the table should be substituted into the equation and the predicted rating calculated. Normalization functions which include several metrics can be used if available, otherwise functions for individual metrics should be used. This predicted rating should be compared with the specified rating.

To illustrate the procedure the normalization function that has been developed for the factor Flexibility will be used. The normalization function, applicable during the design phase, relates measures of modular implementation to the flexibility of the software. The predicted rating of flexibility is in terms of the average time to implement a change in specifications. The normalization function is shown in Figure 4.4-1. The measurements associated with the modular implementation metric are taken from design documents. The measurements involve identifying if input, output and processing functions are mixed in the same module, if application and machine-dependent functions are mixed in the same module and if processing is data volume limited. As an example, assume the measurements were applied during the design phase and a value of 0.65 was measured. Inserting this value in the normalization function results in a predicted rating for flexibility of .33 as identified by point A in Figure 4.4-1. If the Acquisition Manager had specified a rating of 0.2 which is identified by point B, he has an indication that the software development is progressing well with respect to this desired quality.

An organization using this manual is encouraged to establish these functions in its specific environment by following the procedures described in [MCCA77] and [MCCA79].

Table 4.4-1 Normalization Functions

RELIABILITY (DESIGN)		
MULTIVARIATE FUNCTION	$.18\ M_{ET.1} + .19\ M_{SI.3}$	ET.1 Error Tolerance Checklist SI.3 Complexity Measure
INDIVIDUAL FUNCTIONS	$.34\ M_{ET.1}$ $.34\ M_{SI.3}$	
RELIABILITY (IMPLEMENTATION)		
MULTIVARIATE FUNCTION	$.48M_{ET.1} + .14M_{SI.1}$	ET.1 Error Tolerance Checklist SI.3 Complexity Measure SI.1 Design Structure Measure SI.4 Coding Simplicity Measure
INDIVIDUAL	$.57\ M_{ET.1}$ $.58\ M_{SI.1}$ $.53\ M_{SI.3}$ $.53\ M_{SI.4}$	
MAINTAINABILITY (DESIGN)		
INDIVIDUAL FUNCTION	$.57\ M_{SI.3}$ $.53\ M_{SI.1}$	SI.3 Complexity Measure SI.1 Design Structure Measure
MAINTAINABILITY (IMPLEMENTATION)		
MULTIVARIATE FUNCTION	$-.2+.61\ M_{SI.3} + .14M_{MO.2} + .33_{SD.2}$	SI.3 Complexity Measure MO.2 Modular Implementation Measure SD.2 Effectiveness of Comments Measure SD 3 Descriptiveness of Implementation Language Measure SI.1 Design Structure Measure SI.4 Coding Simplicity Measure
INDIVIDUAL FUNCTIONS	$2.1\ M_{SI.3}$ $.71\ M_{SD.2}$ $.6\ M_{SD.3}$ $.5\ M_{SI.1}$ $.4\ M_{SI.4}$	
FLEXIBILITY (DESIGN)		
INDIVIDUAL FUNCTIONS	$.51\ M_{MO.2}$ $.56\ M_{GE.2}$	MO.2 Modular Implementation GE.2 Generality Checklist

(Continued)

Table 4.4-1 Normalization Functions (Continued)

FLEXIBILITY (IMPLEMENTATION)		
MULTIVARIATE FUNCTION	$.22M_{MO.2}+.44M_{GE.2}+.09M_{SD.3}$	
INDIVIDUAL FUNCTIONS	$.6M_{MO.2}$ $.72M_{GE.2}$ $.59M_{SD.2}$ $.56M_{SD.3}$	MO.2 Modular Implementation Measure GE.2 Generality Checklist SD.2 Effectiveness of Comments Measure SD.3 Descriptiveness of Implementation Language Measure
PORTABILITY (IMPLEMENTATION)		
MULTIVARIATE FUNCTION	$-1.7+.19M_{SD.1}+.76M_{SD.2}+2.5M_{SD.3}+.64M_{MI.1}$	
INDIVIDUAL FUNCTIONS	$1.07M_{SI.1}$ $1.1M_{MI.1}$ $1.5M_{SD.2}$	SD.1 Quantity of Comments SD.2 Effectiveness of Comments Measure SD.3 Descriptiveness of Implementation Language Measure MI.1 Machine Independence Measure SI.1 Design Structure Measure

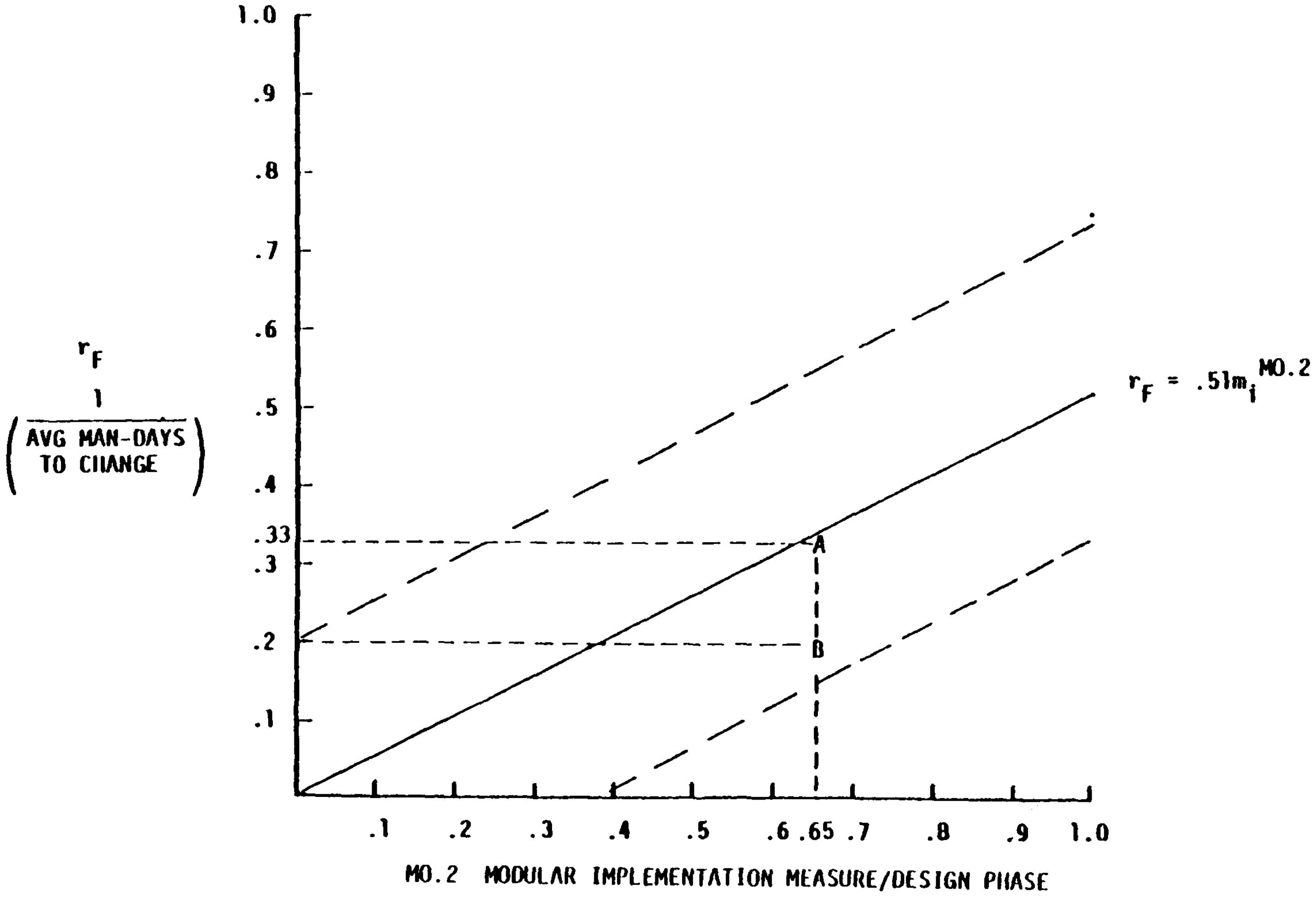
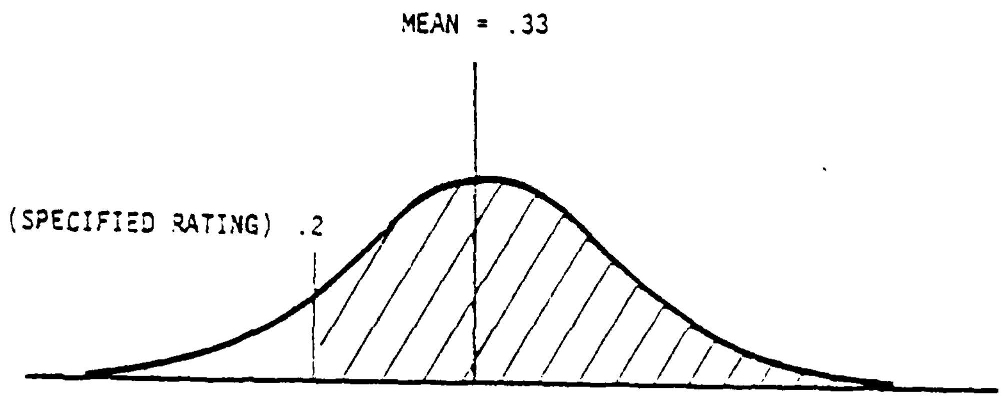


Figure 4.4-1 Normalization Function for Flexibility During Design

3b. *Calculate Confidence in Quality Assessment*

Using statistical techniques a level of confidence can be calculated. The calculation is based on the standard error of estimate for the normalization function (given in Table 3.3.3-2, Vol. I) and can be derived from a normal curve table found in most statistics texts. An example of the devivation process is shown in Figure 4.4-2 for the situation described above. Here it is shown that the Acquisition Manager has an 86 percent level of confidence that the flexibility of the system will be better than the specified rating.



MEAN = .33 (PREDICTED RATING)

STANDARD DEVIATION = .12 (STANDARD ERROR OF ESTIMATE)

LEVEL OF CONFIDENCE = $Pr\left\{x \geq .2\right\}$ = .86 (SHADED AREA)

Figure 4.4-2 Determination of Level of Confidence

4.5 REPORTING ASSESSMENT RESULTS

Each of the preceeding steps described in this section are easily automated. If the metrics are applied automatically then the metric data is available in machine readable form. If the worksheets are applied manually, then the data can be entered into a file, used to calculate the metric, and formatted into the measurement matrix format. The automation of the analyses involve simple matrix manipulations. The results of the analyses should be reported at various levels of detail. The formats of the reports are left to the discretion of the quality assurance organization. The content of the reports to the different managers is recommended in the following paragraphs.

1a. *Report to the Acquisition Manager/Development Manager*

The report content to the Acquisition Manager and the Development manager should provide summary information about the progress of the development toward the quality goals identified at the be beginning of the project.

For example if ratings were specified for several quality factors, the current predicted ratings should be reported.

QUALITY GOALS		PREDICTED RATING BASED ON DESIGN DECUMENT
RELIABILITY	.9	.8
MAINTAINABILITY	.8	.95

If specific ratings were not identified but the important qualities were identified, a report might describe the percentage of modules that currently are judged to be below the average quality (as a result of the sensitivity analysis) or that are below a specified threshold value (as a result of the threshold analysis). These statistics provide a progress status to the manager. Further progress status is indicated by reporting the quality growth of the system or of individual modules. The quality growth is depicted by reporting the scores achieved during the various phases of development. Ultimately the ratings should progressively score higher than those received during requirements. This progress is based on the identification of problems in the early phases which can then be corrected.

1b. *Reports to Quality Assurance Manager*

In addition to the summary quality progress reports described in 1a, the quality assurance manager and his staff will want detailed metric reports. These reports will provide all of the results of the Analyses described in 4.2, 4.3, and 4.4, and perhaps provide the measurement matrix itself for examinations. In addition to the detailed reports, the quality assurance manager should be provided with reports on the status of the application of the metrics themselves by the quality assurance staff. These status reports will provide information on total number of modules and the number which inspectors have analyzed.

1c. *Reports to the Development Team*

The development team should be provided detailed information on an exception basis. This information is derived from the analyses. Examples of the information would be quality problems that have been identified. which characteristics or measurements of the software products are poor, and which modules have been identified as requiring rework. These exception reports should contain the details of why the assessment revealed them as potential problems. It is based onthis information that corrective actions will be taken.

REFERENCES

[MCCA77] McCall, J., Richards, P., Walters, G., "Factors in Software Quality", RADC-TR-77-369, Nov 1977, 3 Vols (A049014)(A049015) & (A049055).

[McCA79] McCall, J., Matsumoto, M., "Metrics Enhancements", RADC-TR-79 , Aug 1979.

[WEIN72] Weinberg,G, "The Psychology of Improved Programming Performance," DATAMATION, Nov 1972.

[CAVA78] Cavano, J, McCall, J, "A Framwork for the Measurement of Software Quality," Proceedings of the ACM Software Quality Assurance Workshop, Nov 1978.

[DUVA76] Duvall, L.M., "Software Data Repository Study," RADC-TR-76-387, Dec 76, (A050636).