AD A049055

2

RADC-TR-77-369, Vol ~~(of three)~~ - 3
Final Technical Report,
~~November 1977~~

11 NOV 77 12 41p.

FACTORS IN SOFTWARE QUALITY, Volume - III, Preliminary Handbook on Software Quality for an Acquisition Manager,

Jim A. McCall
Paul K. Richards
Gene F. Walters

General Electric Company

15 F30602-76-C-0417

16 2237 17 03

Approved for public release; distribution unlimited.

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

149 450

RADC-TR-77-369, Vol III (of three) has been reviewed and approved for publication.

APPROVED: Joseph P. Cavano
JOSEPH P. CAVANO
Project Engineer

APPROVED: Alan R Barnum
ALAN R. BARNUM, Assistant Chief
Information Sciences Division

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

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-77-369, Vol III (of three)	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) FACTORS IN SOFTWARE QUALITY Preliminary Handbook on Software Quality for an Acquisition Manager		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report Aug 76 - Jul 77 6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) Jim A. McCall Paul K. Richards Gene F. Walters		8. CONTRACT OR GRANT NUMBER(s) F30602-76-C-0417
9. PERFORMING ORGANIZATION NAME AND ADDRESS General Electric/Command & Information Systems 450 Persian Drive Sunnyvale CA 94086		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 64740F 22370301
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIS) Griffiss AFB NY 13441		12. REPORT DATE November 1977 13. NUMBER OF PAGES
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)

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

Software Quality
Quality Factors
Metrics
Software Measurements

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

An hierarchical definition of factors affecting software quality was compiled after an extensive literature search. The definition covers the complete range of software development and is broken down into non-oriented and software-oriented characteristics. For the lowest level of the software-oriented factors, metrics were developed that would be independent of the programming language. These measurable criteria were collected and validated using actual Air Force data bases. A handbook was generated that will be useful to Air Force

acquisition managers for specifying the overall quality of a software system.

PREFACE

This document is the final technical report (CDRL A003) for the Factors in Software Quality Study, contract number F030602-76-C-0417. The contract was performed in support of the U.S. Air Force Electronic Systems Division's (ESD) and Rome Air Development Center's (RADC) mission to provide standards and technical guidance to software acquisition managers.

The report consists of three volumes, as follows:

Volume I	Concept and Definitions of Software Quality
Volume II	Metric Data Collection and Validation
Volume III	Preliminary Handbook on Software Quality for an Acquisition Manager

The objective of the study was to establish a concept of software quality and provide an Air Force acquisition manager with a mechanism to quantitatively specify and measure the desired level of quality in a software product. Software metrics provide the mechanism for the quantitative specification and measurement of quality.

This third volume is a preliminary stand-alone reference document to be used by an acquisition manager to implement the techniques established during the study.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION 1
INTRODUCTION

1.1 PURPOSE

In the acquisition of a new software system, a major problem facing a System Program Office (SPO) is to specify the requirements to the software developer and then to determine whether those requirements are being satisifed as the software system evolves. The parameters of the specification center about the technical definition of the application and the function of the software within the overall system. Following this specification, a realistic schedule and costs are negotiated.

There has been no mechanism for specifying the qualities or characteristics of the software - qualities such as reliability, maintainability, usability, testability, and portability. The importance of these software qualities which go beyond the technical mission has been recognized in recent years as a necessary concern for software development managers.

This recognition has come about because of the many instances in which the consequences of not considering software quality has driven total project costs and schedule well beyond initial estimates. It has been found that the costs throughout the total life cycle are more affected by the characteristics of the software system than by the mission-oriented functions performed by the software system. Large-scale software systems have sometimes proven untestable, unmodifiable, and largely unusable by operations personnel because of the characteristics of the software.

While the application functions, cost, and schedule aspects of development can be objectively defined, measured, and assessed throughout the development of the system, the quality desired has historically been definable only in subjective terms. This occurs because the SPO has no quantificable criteria against which to judge the quality of the software until he begins to use the system under operation conditions. This usually leaves the SPO with only two alternatives: to incur increased costs or to back off from the requirements initially desired for the system.

The objectives of this handbook are (1) to describe to the acquisition manager what the software qualities or characteristics are that should be included in a specification, (2) provide a mechanism for objectively specifying the software quality requirements, and (3) introduce a methodology for measuring the level of software quality achieved.

1.2 SCOPE

This handbook is based on the results of a study conducted in support of the U.S. Air Force Electronic Systems Division's (ESD) and Rome Air Development Center's (RADC) mission to provide standards and technical guidance to software acquisition managers. The study represented an initial conceptual investigation of the factors of software quality with a limited sample validation of the concept. Further reasearch and demonstrations of the concept are planned. With this fact in mind, three approaches are described for both specifying and measuring software qualities. The approaches are presented in order of increasing quantification. For both specification and measurement, the first two approaches described are immediately implementable and usable, while the third approach is in its conceptual infancy. Further experience and analysis are required to derive the generally usable quantified relationships required by the third approach.

1.3 RELATIONSHIP OF HANDBOOK TO QUALITY ASSURANCE FUNCTION

The techniques described in this handbook are envisioned as an integral part of an overall quality assurance program. Two important facets of quality assurance are covered in the handbook; software quality specification and software quality evaluation. A review of MIL-STD 483 and 490 provides insight into how these techniques fit into current software development and quality assurance practices.

Appendix I, System Specification, Type A, of MIL-STD 490 covers characteristics (paragraph 3.2) of the system which should be described in the specification. Characteristics such as reliability, maintainability, availability, and interchangeability are mentioned but are oriented toward hardware systems. The software quality factors described in Section 2 of this handbook should be incorporated in this section of a system specification. Appendix VI, Computer

Program Development Specifications, of MIL-STD 490 and Appendix VI, Computer Program Configuration Item Specification, of MIL-STD 483 also relate to the specification of software in Section 3, Requirements. It is stated that the requirements "shall contain performance and design requirements" and "specify design constraints and standards necessary to assure compatibility of the CPCI." The design requirements and standards should also include those features that enhance software quality. Section 2 of this handbook provides the terminology and procedures for including requirements for software quality in the specification.

With respect to the measurement of software quality, Section 4 of the aforementioned appendices of MIL-STDs 483 and 490 cover quality assurance provisions. Current emphasis in these sections is on functional testing, the "formal verification of the performance of the CPCI in accordance with the requirements." These quality assurance provisions should also include evaluation of the software quality. Section 3 of this handbook describes approaches utilizing software metrics which quantify and formalize the software quality evaluation procedure.

1.4 BENEFITS OF APPROACH

The concepts of software quality metrics expressed in this handbook contribute to a more disciplined, engineering approach to software quality assurance. The software acquisition manager is provided with conceptually simple, easy to use procedures for specifying required quality in more precise terminology. The specification procedures force a life-cycle view at the initial planning stages of a software development. An acquisition manager essentially performs a tradeoff analysis between the quality factors, cost, and schedule in establishing the requirements of the system. The software developer is forced to address how they plan to build the required quality into the software.

Specific software quality attributes required are independent of the design and implementation techniques used by the software developer. Further insurance or confidence is gained from this more precise description of the quality requirements.

The software quality metrics provide a more formal, consistent means of evaluating/inspecting the software products developed during a software system development effort. They also provide a means of tracking the progress since they are applied at several points in time during the development. The consistency provides a basis for comparison between projects or between different data collectors.

The metric indicator concept adds an additional dimension in that it provides a mechanism for pinpointing difficiencies. Evaluation is required to determine what corrective action, if any, need be taken.

Many of the metrics can be automatically collected enhancing their accuracy and consistency. The metrics are applied during all phases with increasingly greater confidence in their indication of quality. The accumulation of the metric data and the application of these analytical techniques enhance the understanding of the software development process and the characteristics of good design and implementation.

The cumulative effect of these procedures and techniques is that a quantitative relationship between metrics and software quality ratings can be derived and used to predict how well the software development is progressing toward achieving the required quality in the resulting software product. This ability to specify software quality and quantitatively measure the development in terms of the quality will assist the acquisition manager and software developer in producing higher quality products.

1.5 HANDBOOK ORGANIZATION

The handbook has been organized as a reference document for an acquisition manager, describing the concepts of software quality and identifying where more detailed information can be found.

The first section provides introductory information, definitions, and recommendations for use of the handbook.

The second section describes three approaches to specifying software quality. Each description is organized in the following manner:

- General discussion of approach
- Steps to be followed

The third section describes three approaches to measuring software quality. Each approach is described using the same format of the second section.

The three approaches in each section are presented in order of increasing formality and quantification of the relationship between the metrics and the quality factors. The approaches are titled and can be easily identified in the handbook as shown in Table 1-1.

1.6 RELATIONSHIP OF HANDBOOK TO FACTORS IN SOFTWARE QUALITY FINAL REPORT

This handbook is based on and utilizes the concepts described in the Factors in Software Quality Final Report, Volumes I and II, and previous research efforts related to software quality referenced in that report. The report is the result of work performed for ESD and RADC under contract number F030602-76-C-0417.

It is recommended that the Factors in Software Quality Final Report be read and used as a supporting reference for implementing the concepts described in this handbook.

It is also recommended that the metric table (Table 6.2.1) contained in the first volume of that report be used as a data collection form in applying the metrics.

Table 1-1 Presentation of Approaches to Specifying and Measuring Software Quality

TOPIC	APPROACH #	PRESENTED IN PARAGRAPH #	TITLE
SPECIFYING SOFTWARE QUALITY	1	2.2	SPECIFICATIONS USING SOFTWARE QUALITY FACTORS
	2	2.3	IDENTIFICATION OF CRITICAL SOFTWARE ATTRIBUTES
	3	2.4	QUANTIFICATION OF SOFTWARE QUALITY
MEASURING SOFTWARE QUALITY	1	3.2	*FORMAL INSPECTION OF SOFTWARE PRODUCTS USING METRICS*
	2	3.3	METRIC INDICATOR CONCEPT
	3	3.4	FORMAL RELATIONSHIP OF METRICS TO QUALITY FACTORS

1.7 DEFINITIONS

The following definitions are provided as explanation of terminology used in this handbook:

- Software: the programs and documentation associated with and resulting from the software development process.
- Quality: a general term applicable to any trait or characteristic, whether individual or generic; a distinguishing attribute which indicates a degree of excellence or identifies the basic nature of something.
- Factor: a condition or characteristic which actively contributes to the quality of the software. For standardization purposes, all factors will be related to a normalized cost to either perform the activity characterized by the factor or to operate with that degree of quality. For example, maintainability is the effort required to locate and fix an error in an operational program. This effort required may be expressed in units such as time, dollars, or manpower. The following rules apply to the set of software quality factors:
  - A condition or characteristic which contributes to software quality
  - A user-related characteristic
  - Related to cost either to perform the activity characterized by the function or to operate with that degree of quality
  - Relative characteristic between software products

  The last rule, that a factor is a relative characteristic between software products, requires a brief explanation. Figure 1-1 illustrates the relationship between a factor and the cost to achieve different levels of that quality factor. As an example, assume the curve describes the cost versus level of quality relationship for the factor reliability. A much lower level of reliability, which costs less to achieve, may be as acceptable to a management information system (MIS) acquisition manager as a much higher level is to a command and control (C^2) manager due to the nature of the individual applications. So, while the C^2 final product may have a higher degree of reliability according to our measures, it may be no more acceptable to the user than the MIS system with its lower reliability is to its user.

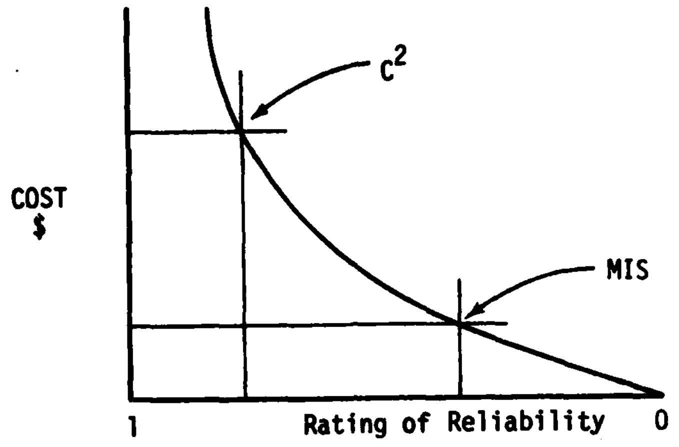


Figure 1-1 Relationship of Software Quality to Cost

- <u>Criteria</u>: attributes of the software or software production process by which the factors can be judged and defined. The following rules apply to the criteria:
    - Attributes of the software or software products of the development process; i.e., criteria are software-oriented while factors are user-oriented
    - May display a hierarchical relationship with subcriteria
    - May affect more than one factor
- <u>Metrics</u>: quantitative measures of the software attributes related to the quality factors. The measures may be objective or subjective. The units of the metrics are chosen as the ratio of actual occurrences to the possible number of occurrences.
- <u>Normalization Function</u>: a formal mathematical relationship between a set of metrics and a rating of a quality factor.

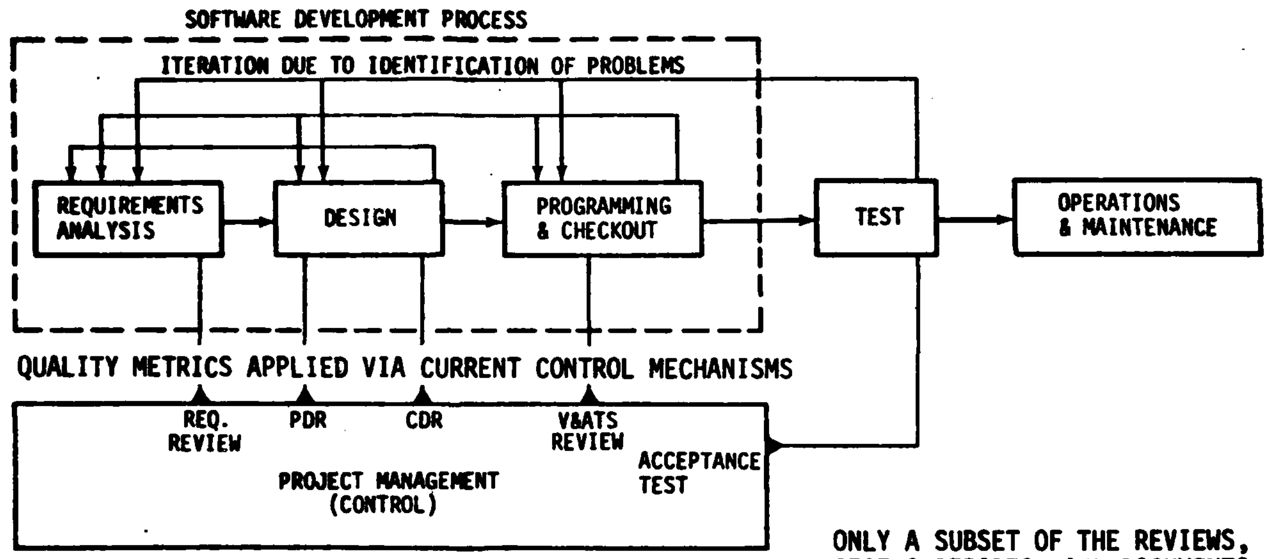
1.8 RECOMMENDED USE OF HANDBOOK

It is recommended that this handbook be used by an acquisition manager as a reference during the preparation of a software system specification, or a request for proposal for a software system development, and during evaluation activities of the software development phase.

The concept of quality metrics as described in this handbook involves the application of the metrics via already existing control mechanisms, i.e., reviews, status reports, documentation delivered during the development, and source code. The current emphasis of these controls is to evaluate the schedule and cost performance and to determine the functional correctness of the software being developed. The quality metrics are applied to these control vehicles to provide an indication of the quality of the software product being developed. This concept is illustrated in Figure 1-2.

The metric data collection (measuring) can be done by the contractor or developer and reported during reviews or it can be performed by the acquisition manager's quality assurance or verification and validation personnel.

The order in which the three approaches to specifying and measuring software quality are presented not only corresponds to the increasing formality of the relationship between quality and the metrics but also represents an increasing manpower requirement for implementation and an increasing requirement for experience on the part of the SPO with the concepts. Based on these facts, it is recommended that the concepts be incrementally phased into an acquisition manager's operation. The first approach described, for both specifying and measuring quality, can be implemented with a minimum amount of effort and will yield immediate benefits to the acquisition manager. The second approach can then be phased in, requiring additional effort, but providing higher confidence in the indication of the level of quality being achieved during the development process. The phased approach to the introduction of the quality metrics concepts allows for training, familiarization, and experience to be gained while the concepts are actually used and benefits realized. The third approach will become more viable as experience and historical data is accumulated using the first two approaches. The mathematical relationships required by the third approach are derived from the historical data.


SOFTWARE DEVELOPMENT PROCESS
ITERATION DUE TO IDENTIFICATION OF PROBLEMS
REQUIREMENTS ANALYSIS
DESIGN
PROGRAMMING & CHECKOUT
TEST
OPERATIONS & MAINTENANCE
QUALITY METRICS APPLIED VIA CURRENT CONTROL MECHANISMS
REQ. REVIEW
PDR
CDR
V&ATS REVIEW
ACCEPTANCE TEST
PROJECT MANAGEMENT (CONTROL)

LEGEND	
REQ REVIEW	Requirements Review
PDR	Preliminary Design Review
CDR	Critical Design Review
V&ATS REVIEW	Validation and Acceptance Test Specification Review

ONLY A SUBSET OF THE REVIEWS, STATUS REPORTS, AND DOCUMENTS PROVIDED DURING A SOFTWARE DEVELOPMENT ARE SHOWN.

1604-2

Figure 1-2 Software Development Process Control

SECTION 2

SPECIFYING SOFTWARE QUALITY

2.1 CONCEPT OF FACTORS IN SOFTWARE QUALITY

An acquisition manager's involvement with a software product can be categorized in terms of three distinct activities as follows:

SOFTWARE PRODUCT ACTIVITIES

- Product Operation
- Product Revision
- Product Transition

Specific qualities or characteristics (quality factors) of the software product are related to these activities as shown in Figure 2-1. The questions in parentheses provide the relevancy or interpretation of the factors to an acquisition manager.

Thus, with respect to the operation of a software system, an acquisition manager's concern for quality is in terms of its correctness, *reliability*, *efficiency*, integrity, and usability. Over the life cycle of a system, revisions to the system may be necessary due to problems or changing requirements. The acquisition manager is therefore concerned with the maintainability, flexibility, and testability of the software. Longer range considerations may involve moving the software to another hardware system, interfacing it with another system, or developing newer versions of the system. The related quality concerns are for portability, interoperability, and reusability.

The definitions of these eleven quality factors are in Table 2-1.

All of the quality factors should be considered in the initial specification for the software. The first approach (paragraph 2.2) describes the procedure for considering these quality factors. The following paragraphs (2.3, 2.4) describe progressively more detailed approaches to specifying software quality.

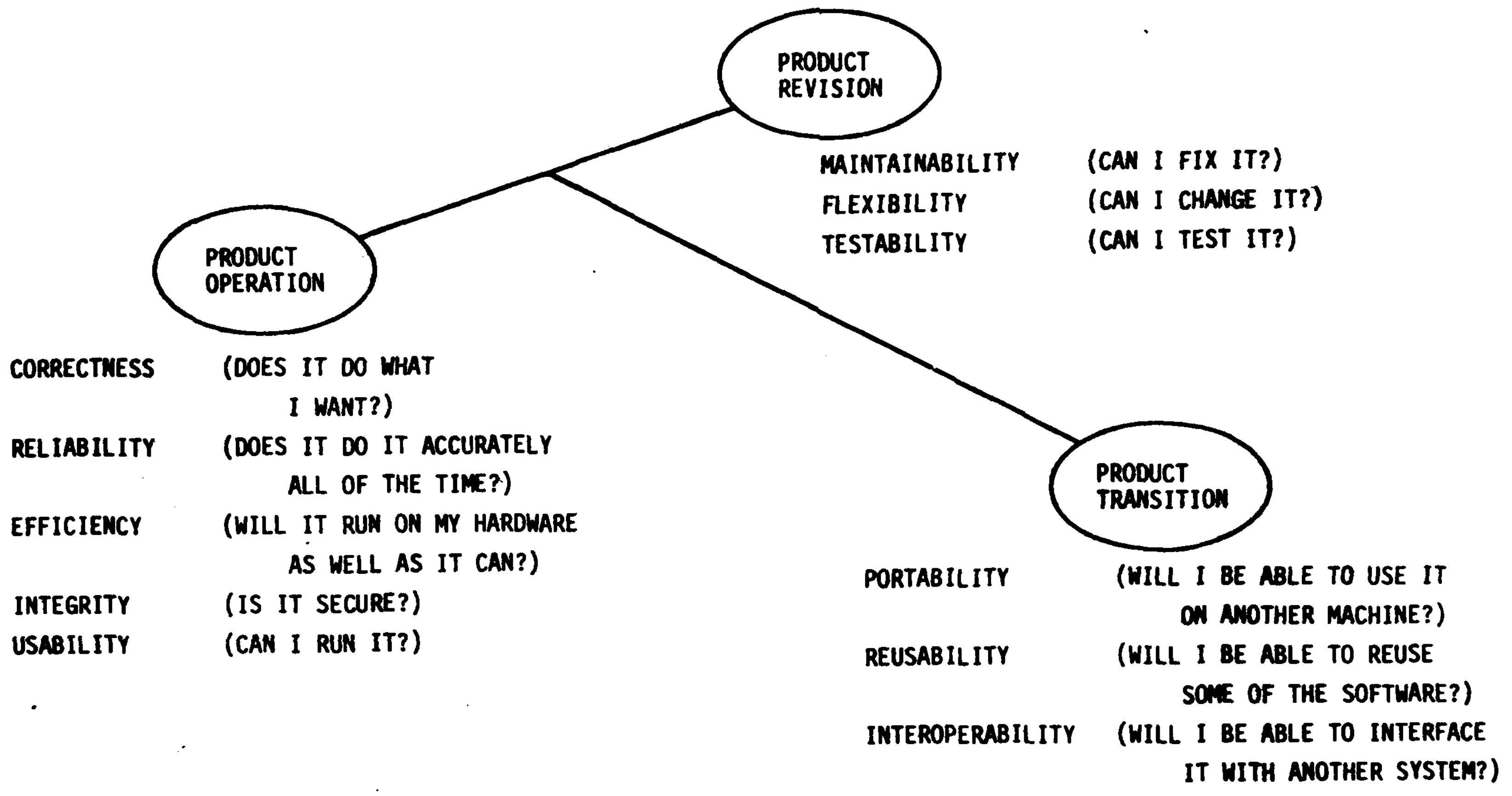

PRODUCT REVISION
MAINTAINABILITY (CAN I FIX IT?)
FLEXIBILITY (CAN I CHANGE IT?)
TESTABILITY (CAN I TEST IT?)
PRODUCT OPERATION
CORRECTNESS (DOES IT DO WHAT I WANT?)
RELIABILITY (DOES IT DO IT ACCURATELY ALL OF THE TIME?)
EFFICIENCY (WILL IT RUN ON MY HARDWARE AS WELL AS IT CAN?)
INTEGRITY (IS IT SECURE?)
USABILITY (CAN I RUN IT?)
PRODUCT TRANSITION
PORTABILITY (WILL I BE ABLE TO USE IT ON ANOTHER MACHINE?)
REUSABILITY (WILL I BE ABLE TO REUSE SOME OF THE SOFTWARE?)
INTEROPERABILITY (WILL I BE ABLE TO INTERFACE IT WITH ANOTHER SYSTEM?)

Figure 2-1 Allocation of Software Quality Factors to Product Activity

1352

Table 2-1 Definition of Software Quality Factors

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.

2.2 SPECIFICATIONS USING SOFTWARE QUALITY FACTORS

2.2.1 DISCUSSION OF FIRST APPROACH

This first approach to specifying software quality uses the conceptualization of factors in software quality described in the previous paragraph as the basic mechanism for the acquisition manager to identify requirements for quality in a software product which have complete life-cycle implications. For example, if the SPO is sponsoring the development of a system 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, reusability 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 and the concept of interoperability is extremely important. All of these considerations can be accommodated in the framework derived.

2.2.2 STEPS TO BE FOLLOWED

1. In preparing a request for proposal (RFP) or system requirements specification (SRS), the acquisition manager should identify and assign priorities to the critical quality factors.

   Each software system is unique in its software quality requirements. There is no specific categorization of applications which can be related to certain levels of quality. There are certain basic system characteristics which effect the quality requirements. Each system must be evaluated for its fundamental characteristics. These fundamental characteristics and the related quality factors are identified in Table 2-2.

Table 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

These basic characteristics should be taken into account when the critical quality factors are identified.

In considering all of the quality factors, the life-cycle implications of the system are considered. Table 2-3 identifies the life-cycle implications (impact of poor quality) of the quality factors and should be used as an input in identifying the relative importance of the quality factors.

During the process of identifying the importance of the quality factors, the tradeoffs between certain quality factors should be recognized. Table 2-4 should be utilized as a guide for determining the conflicts in quality requirements. A few examples are provided to illustrate how the table is interpreted:

<u>Maintainability vs Efficiency</u> - 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.

<u>Integrity vs Efficiency</u> - the additional code and processing required to control the access of the software or data usually lengthen run time and require additional storage.

<u>Interoperability vs Integrity</u> - 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.

Table 2-3 The Impact of not Specifying or Measuring Software Quality Factors

	DEVELOPMENT			EVALUATION	OPERATION		
FACTORS \ LIFE-CYCLE PHASES	REQMTS ANALYSIS	DESIGN	CODE & DEBUG	SYSTEM TEST	OPERATION	MAINTENANCE	TRANSITION
CORRECTNESS	△	△	△	X	X	X	
RELIABILITY	△	△	△	X	X	X	
EFFICIENCY		△	△		X		
INTEGRITY	△	△	△		X		
USABILITY	△	△		X	X	X	
MAINTAINABILITY		△	△			X	X
TESTABILITY		△	△	X		X	X
FLEXIBILITY		△	△			X	X
PORTABILITY		△	△				X
REUSABILITY		△	△				X
INTEROPERABILITY		△			X		X

LEGEND

△ - where quality factors should be measured

X - where impact of poor quality is realized

Table 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

It should be recognized this table only provides general guidelines and further analyses along these lines should be made for specific cases. It is important to note that if a high level of quality is required for conflicting factors, the cost to achieve the requirements may be very high.

2. Once the critical quality factors have been identified and priorities assigned, they should be included in the RFP or SRS with definitions from paragraph 2.1, and the developer required to comment on how the software will be developed to exhibit the qualities specified.

3. Wherever possible, as much detailed explanation should be included with the definition for each quality factor. For example, if portability is a major concern to an acquisition manager, as precise a description as possible should be included as to the types of environments to which the system might be transported.

2.3 IDENTIFICATION OF CRITICAL SOFTWARE ATTRIBUTES

2.3.1 DISCUSSION OF SECOND APPROACH

This approach involves a refinement to the first approach described in paragraph 2.2. Each quality factor is further defined by criteria which are the software attributes whose presence in the software enhances the characteristic represented by the quality factor.

The criteria identified in the Factors in Software Quality Final Report are shown in Figure 2-2, indicating which quality factors they significantly impact, and are defined in Table 2-5.

These criteria are used in this approach to further define the quality requirements.

2.3.2 STEPS TO BE FOLLOWED

1. Having identified the critical quality factors, the acquisition manager then identifies the related critical software attributes which are required. For example, if the acquisition manager wants

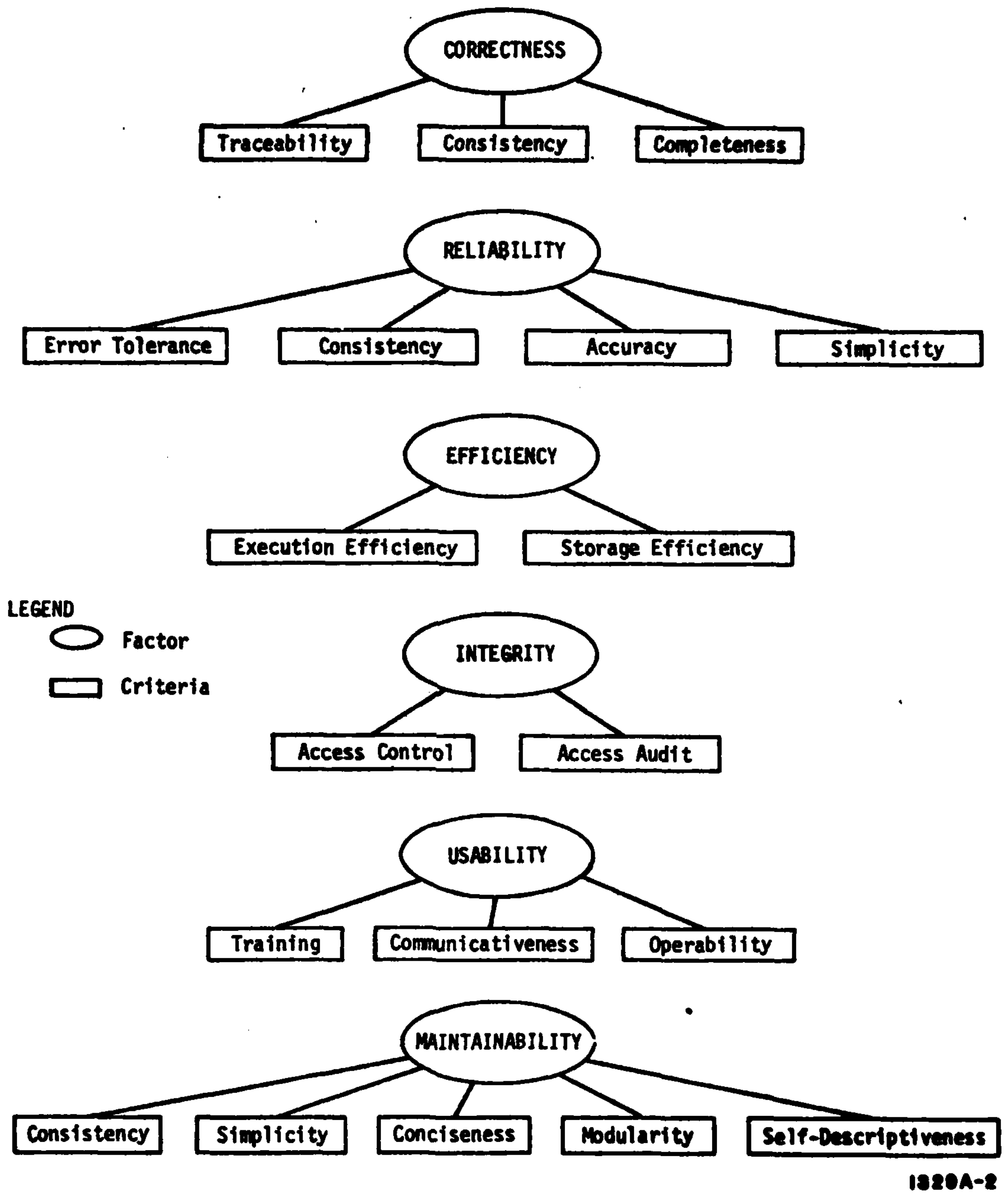


Figure 2-2 Relationship of Criteria to Software Quality Factors

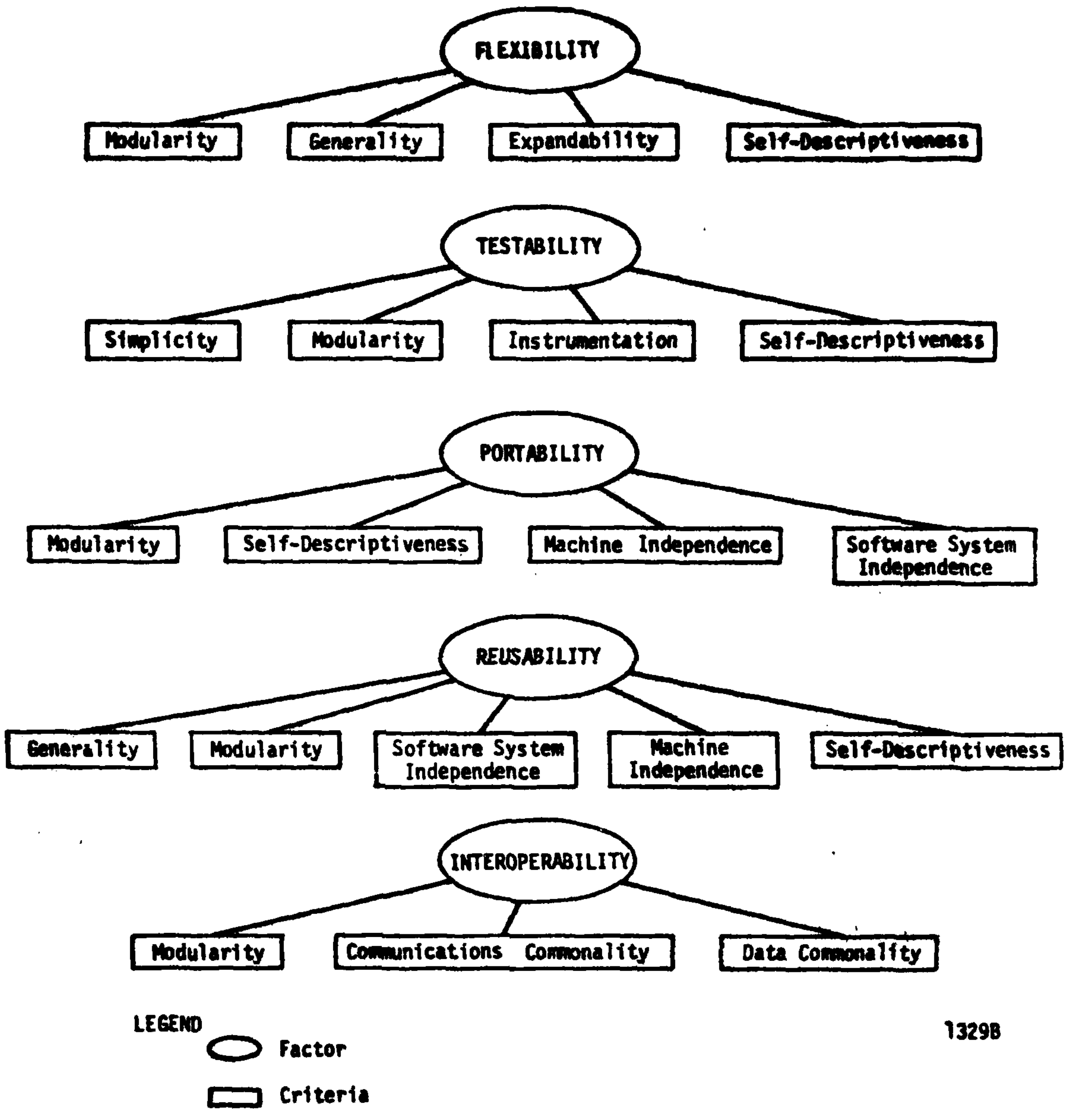


Figure 2-2 Relationship of Criteria to Software Quality Factors (continued)

Table 2-5 Criteria Definitions for Software Quality Factors

CRITERION	DEFINITION	RELATED FACTORS
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.	Correctness
COMPLETENESS	Those attributes of the software that provide full implementation of the functions required.	Correctness
CONSISTENCY	Those attributes of the software that provide uniform design and implementation techniques and notation.	Correctness Reliability Maintainability
ACCURACY	Those attributes of the software that provide the required precision in calculations and outputs.	Reliability
ERROR TOLERANCE	Those attributes of the software that provide continuity of operation under nonnominal conditions.	Reliability
SIMPLICITY	Those attributes of the software that provide implementation of functions in the most understandable manner. (Usually avoidance of practices which increase complexity.)	Reliability Maintainability Testability
MODULARITY	Those attributes of the software that provide a structure of highly independent modules.	Maintainability Flexibility Testability Portability Reusability Interoperability
GENERALITY	Those attributes of the software that provide breadth to the functions performed.	Flexibility Reusability
EXPANDABILITY	Those attributes of the software that provide for expansion of data storage requirements or computational functions.	Flexibility
INSTRUMENTATION	Those attributes of the software that provide for the measurement of usage or identification of errors.	Testability
SELF-DESCRIPTIVENESS	Those attributes of the software that provide explanation of the implementation of a function.	Flexibility Maintainability Testability Portability Reusability

Table 2-5 Criteria Definitions for Software Quality Factors (continued)

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

to stress the importance of maintainability, the following software attributes would be identified as required in the RFP or SRS:

Consistency
Simplicity
Conciseness
Modularity
Self-Descriptiveness

The precise definitions of these attributes would be included in the RFP or SRS.

2. The acquisition manager should evaluate the software developer's plan to provide the required software attributes for each quality factor.

2.4 QUANTIFICATION OF SOFTWARE QUALITY

2.4.1 DISCUSSION OF THIRD APPROACH

This approach requires precise statements of the level of quality required of the software. Currently, the underlying mathematical relationships which allow measurement at this level of precision do not exist for general use. The procedures for developing those relationships are documented in the Factors in Software Quality Final Report. The mechanism for making the precise statement for any quality factor is a rating of that factor. The ratings are explained in Table 2-6.

2.4.2 STEPS TO BE FOLLOWED

1. After identification of the critical quality factors, specific performance levels or ratings required for each factor should be specified. For example, a 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 in the RFP. To comply with this

Table 2-6 Problem Report and Man-Power Expenditure Categorization

CATEGORY BY QUALITY FACTOR	EXPLANATION
• CORRECTNESS	The function which the software is to perform is incorrect. The rating is in terms of effort required to fix.
• RELIABILITY	The software does not function as expected. The rating is in terms of effort required to fix.
• EFFICIENCY	The software does not meet performance (speed, storage) requirements. The rating is in terms of effort required to fix.
• INTEGRITY	The software does not provide required security. The rating is in terms of effort required to fix.
• 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 fix.
• MAINTAINABILITY	The rating is in terms of effort required to correct any of the above problems.
• FLEXIBILITY	The rating is in terms of effort required to make a modification due to a change in specifications.
• TESTABILITY	The rating is in terms of effort required to test changes or fixes.
• REUSABILITY	The rating is in terms of effort required to use software in a different application.
• PORTABILITY	The rating is in terms of effort required to convert the software to operate in a different environment.
• INTEROPERABILITY	The rating is in terms of effort required to couple the system to another system.

specification, the software would have to exhibit characteristics which, when present, give an indication that the software will perform to this rating. These characteristics are measured by metrics. The measurements are inserted in a mathematical relationship and a predicted rating is obtained.

2. The specific metrics should be identified which will be applied to various software products of the development phase to provide an indication of the progress toward achieving the required level of quality. These metrics will be discussed further in the next section and are defined in Section 6 of the Factors in Software Quality Final Report.

SECTION 3

MEASURING SOFTWARE QUALITY

3.1 THE CONCEPT OF QUALITY METRICS

Figure 3-1 illustrates the concept of applying metrics during the development of a software system. The metrics are quantitative measures of the software attributes (criteria identified in paragraph 2.3) which are necessary to realize certain characteristics (quality factors) in the software. The metrics provide an indication of the progression toward the achievement of high quality end products. Specific acceptance tests can be oriented toward evaluating the levels of quality achieved but these testing strategies are not within the scope of this handbook.

As previously mentioned, the metrics have been developed to be applied to products currently provided during a software development. They may be applied either by acquisition manager personnel to delivered products, by contractor personnel and reported in summary format to the acquisition manager during reviews, or by contractor personnel as part of their own quality assurance program.

The metrics were developed so as not to restrict or interfere with the management and development methodologies and techniques of the developer.

The metrics are listed in Table 6.2-1 of the Factors in Software Quality Final Report with definitions following that table. Their application to software products is described in Appendix D of that report. Typical automated tools available in software development environments which assist in the metric data collection are identified in Section 8 of that report also.

For illustration, some examples of metrics are provided in Table 3-1. The complexity measure is calculated from a design chart and from source code utilizing path flow analysis and data set/use information. The effectiveness of comments measure is a quantitative measure based on objective guidelines for inspecting the source code for the existence or absence of comments at

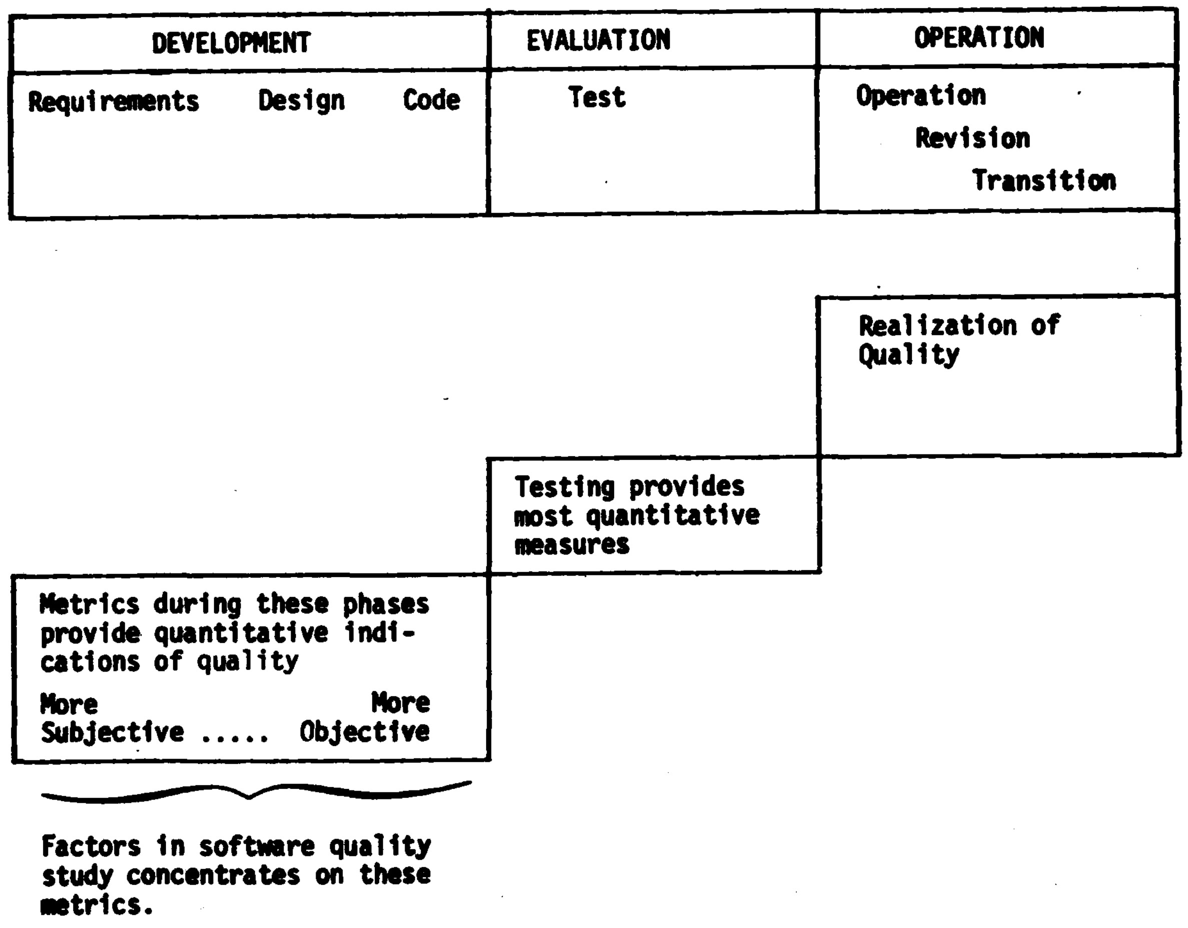

DEVELOPMENT
EVALUATION
OPERATION
Requirements
Design
Code
Test
Operation
Revision
Transition
Realization of Quality
Testing provides most quantitative measures
Metrics during these phases provide quantitative indications of quality
More Subjective ..... More Objective
Factors in software quality study concentrates on these metrics.

Figure 3-1 Concept of Metrics

Table 3-1 Example Metrics

METRIC	MNEMONIC	CRITERION	RELATED QUALITY FACTORS
Data and Control Flow Complexity Measure	SI.3	Simplicity	Reliability Maintainability Testability
Effectiveness of Comments Measure	SD.2	Self-Descriptiveness	Flexibility Maintainability Reusability Portability
Machine Independence Measure	MI.1	Machine Independence	Portability Reusability
Completeness Checklist	CP.1	Completeness	Correctness

specific locations such as prologue comments with certain information, branching statements, machine dependent code, declarative statements, and so forth. The machine independence measure is a compilation of a number of measurements which indicate the degree of independence of the design and code. The completeness checklist measures attributes that should be included in specifications, design documents, and the code, which, if missing, increase the probability that the final product will be functionally incomplete.

The following paragraphs describe increasingly more detailed approaches to utilizing these metrics to provide an indication of the quality of the software being developed.

3.2 FORMAL INSPECTION OF SOFTWARE PRODUCTS USING METRICS

3.2.1 DISCUSSION OF APPROACH

The first level of measuring software quality involves applying the metrics to software products as they are produced. Different sets of metrics are applicable to products produced during the requirements analysis, design, and coding phases of development. The use of the metrics in this manner insures a formal and consistent review of each of the software products.

3.2.2 STEPS TO BE FOLLOWED

1. The subset of metrics which relate to the identified critical quality factors and software attributes and are applicable to the phase of development should be applied to the available software products. For example, during the design phase, metrics could be applied to design specifications, interface control documents, test plans, minutes and materials prepared for reviews, and so on.

2. A subjective evaluation of how well the software is being developed with respect to the specific quality factors can be made based on the inspection of the software products using the metrics.

3.3 METRIC INDICATOR CONCEPT

3.3.1 DISCUSSION OF APPROACH

The second approach 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.

3.3.2 STEPS TO BE FOLLOWED

1. After the metrics are applied to the available software products, the values are obtained and evaluated. If particular modules receive low metric scores, they can be individually evaluated for potential problems. If low metric scores are realized across the system, an evaluation should be made to identify the cause. It may be that a design or implementation technique used widely by the development team is the cause. Corrective action such as the enforcement of a development standard can then be introduced.

2. Further analysis can be conducted. An examination of the metric scores for each module in a system will reveal which metrics vary widely. Further examination will reveal if this variation correlates with the number of problem reports or with historical variances in performance. This sensitivity analysis identifies characteristics of the software, represented by the metrics, which are critical to the quality of the product. Quality assurance personnel should place increased emphasis on these aspects of the software product.

3. Threshold values may be established below which certain actions would be required. A simple example is the percent of comments per line of source code. Certainly code which exhibits only 1% or 2% measurements for this metric would be identified for corrective action. It may be that 10% to 20% is a more industry-wide acceptable level.

3.4 FORMAL RELATIONSHIP OF METRICS TO QUALITY FACTORS

3.4.1 DISCUSSION OF APPROACH

This approach is the most detailed of the three approaches to measuring software quality. The underlying mathematical foundations to the derivation of the relationships are described in Section 7 and Appendix C of the Factors in Software Quality Final Report. Basically, the measurements (m_i) for a subset of metrics are applied to the software products of a specific phase ($\emptyset$) in the software development. When inserted into the corresponding equation (normalization function) a rating for a particular quality factor (r_F) can be predicted as shown below:

$$f^{\emptyset}(m_1, m_2, \ldots, m_k) = r_F$$

Currently, generally applicable predictive equations are not available. Specific normalization functions were developed during the study which resulted in this handbook. They are based on a limited sample and are not recommended for general use. The procedure for the derivation of equations which would be very useful in a partcilar development environment is described in detail within the report.

3.4.2 STEPS TO BE FOLLOWED

1. To illustrate the procedures involved in this approach, a normalization function for the quality factor flexibility developed during the Factors in Software Quality study will be used. The normalization function, applicable during the design phase, relates measures of modular implementation ($MO.2_F$) 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 3-2.

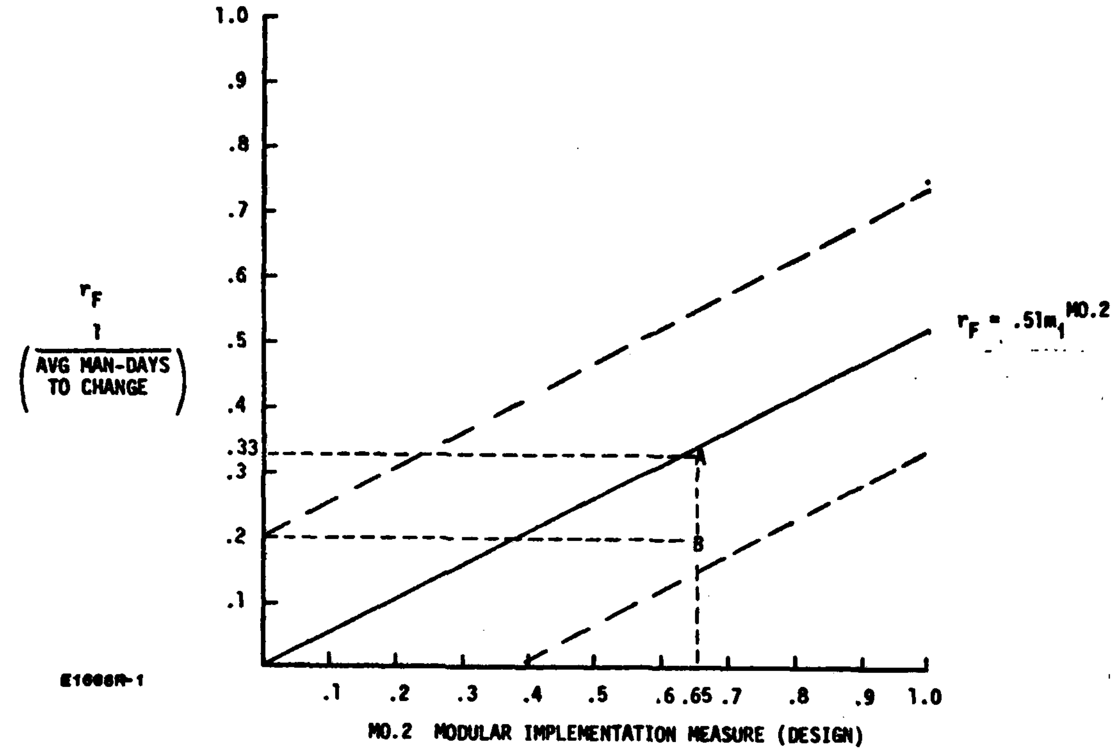


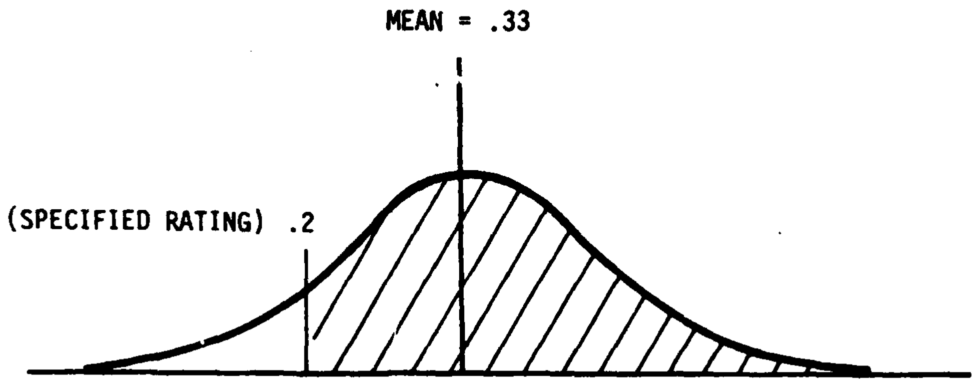
Figure 3-2 Normalization Function (Design) for Flexibility

The measurements associated with $m_i^{MO.2}$ are taken from design documents and reveal 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 or data value limited.

As an example, assume the measurements were applied during the design phase of a project and a metric value of .65 was measured. Inserting this value in the normalization function:

$$r_F = .51\ m_i^{MO.2}$$

results in a predicted rating for flexibility of .33 (identified by point A in Figure 3-2). If the acquisition manager had specified a rating of .2 (1/5 average man-days to change), which is identified by point B in Figure 3-2, he has an indication that the software development is progressing well with respect to this desired quality. By analyzing the variance associated with this normalization function, it is shown in Figure 3-3 that the acquisition manager has an 86% level of confidence that the flexibility of his system will be better than his specified rating.



MEAN = .33 (PREDICTED RATING)
STANDARD DEVIATION = .12 (STANDARD ERROR OF ESTIMATE)
LEVEL OF CONFIDENCE = $\Pr\{x \geq .2\}$ = .86 (SHADED AREA)

Figure 3-3 Determination of Level of Confidence

2. The comparison of the predicted rating with the specified rating provides a more quantitative indication, with an associated level of confidence, of how well the software development is progressing toward achieving the specified levels of quality. Corrective action based on further analysis would be in order if the predicted rating was lower than the specified rating.

MISSION
of
Rome Air Development Center

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

Printed by
United States Air Force
Hanscom AFB, Mass. 01731