RADC-TR-77-369, Volume I (of three)
Final Technical Report
November 1977

FACTORS IN SOFTWARE QUALITY
Concept and Definitions of Software Quality

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

General Electric Company

Approved for public release; distribution unlimited.

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

This report has been reviewed by the RADC Information Office (OI) 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-77-369, Vol I (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

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.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-77-369 Vol I	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): FACTORS IN SOFTWARE QUALITY. Volume I. Concepts and Definitions of Software Quality.		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, 12 03
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIS), Griffiss AFB NY 13441		12. REPORT DATE: November 1977; 13. NUMBER OF PAGES: 158
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

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

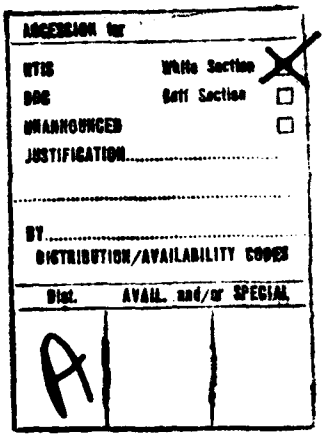
UNCLASSIFIED

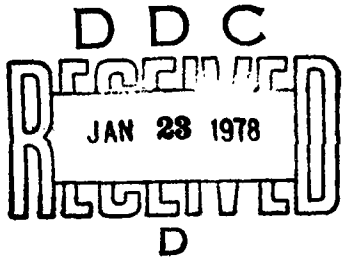
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

149450

GB

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 was prepared by J. McCall, P. Richards, and G. Walters of the Sunnyvale Operations, Information Systems Programs, General Electric Company. Significant contributions were made by A. Breda, S. Reiss, and R. Colenso.

Technical guidance was provided by J. Cavano, RADC Project Engineer and Captain A. French, ESD Technical Monitor.

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 first volume describes the process of developing our concept of software quality and what the underlying software attributes are that provide the quality, and defines the metrics which provide a measure of the degree to which the attributes exist.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

EVALUATION

The Air Force is constantly striving to improve the quality of its software systems. Producing high quality software is a prerequisite for satisfying the stringent reliability and error-free requirements of command and control software. To help accomplish this, a more precise definition of software quality is needed as well as a way to derive metrics for quantifying software for objective analysis. This effort was initiated in response to the need to better understand those factors affecting software quality and fits into the goals of RADC TPO No. 5, Software Cost Reduction in the area of Software Quality (Metrics). General Electric classified over the complete range of software development both user-oriented and software-oriented characteristics which were related to Air Force applications and life-cycle phases. Programming-language independent metrics were defined using Air Force data bases. Finally, formal methodology for the validation of the metrics was developed and used.

The significance of this work is that through the establishment of quality measurement a beneficial impact will occur on the evaluation and implementation of a software product at each stage of development. Trade-offs between technical value and cost will be more easily understood. In addition, Air Force acquisition managers, with the aid of a handbook delivered as part of this contract, will be able to specify requirements to software developers more completely and then determine whether those requirements are being satisfied early enough for corrective action. As quality measurement becomes more vigorous in the future, the Air Force will be capable of establishing software product and service standards for itself and its contractors.

Joseph P. Cavano

JOSEPH P. CAVANO
Project Engineer

SECTION 1

INTRODUCTION/EXECUTIVE SUMMARY

1.1 TASK OVERVIEW

The Factors in Software Quality task was 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. ESD sponsored the task and RADC provided technical project management.

The impetus for this effort and other related work in the analysis of software quality can be traced to recommendations for such research made jointly by DOD, industry, and university representatives at the Symposium on the High Cost of Software [WULFW73] in September, 1973, at the Joint Logistics Commanders Electronic Systems Reliability Workshop (by members of the Software Reliability Working Group) [FIND75] in May, 1975, and more recently by the DOD R&D Panel.

1.2 TASK OBJECTIVES

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 satisfied as the software system evolves. The parameters of the specification center about the technical definition of the application and the software role within the overall system. Following this, a realistic schedule and costs are negotiated.

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 quantifiable criteria against which to judge the quality of the software until he begins to use the system under operational conditions.

As represented in Figure 1.2-1, the objective of this study was to provide guidelines in how to objectively specify the desired amount of quality at the system requirements specification phase. By levying measurable quality criteria on the developer, the SPO will be able to subsequently evaluate the quality of the software not only when the system becomes operational, but also as each phase of the project is completed. As a result of corrective actions the SPO may choose to invoke, these early measurements can significantly reduce impact on life cycle cost and schedule.

The figures drawn with solid lines represent the questions the SPO can now ask and can obtain objective answers. The figures drawn with dashed lines represent areas which cannot presently be addressed. The objective of this task was to provide the mechanism to answer the question of how good the software is more precisely and earlier in the life cycle. The results of this task provide the basis for the SPO to specify and evaluate the software quality quantitatively, as is illustrated with the dash-lined figures.

The approach taken to quantify software quality is summarized as follows:

1. Determine a set of quality factors which jointly comprise software quality. (Section 2,3)
2. Develop a working, hierarchical definition by identifying a set of criteria for each factor. (Section 4)
3. Define metrics for each criterion and a normalization function which relates and integrates the metrics for all of the criteria of a factor to an overall rating of that factor. A scaling of the metrics' contributions to this rating will result in a figure of merit for each factor. (Section 5,6,7)
4. Validate the metrics and normalization functions by utilizing the historical data of two Air Force systems. (Section 7,8)
5. Translate the results of this effort into guidelines that can be used by Air Force Program Offices to specify the quality of the software product required and to measure to determine if the development effort is leading toward that level of quality. (Volume III)

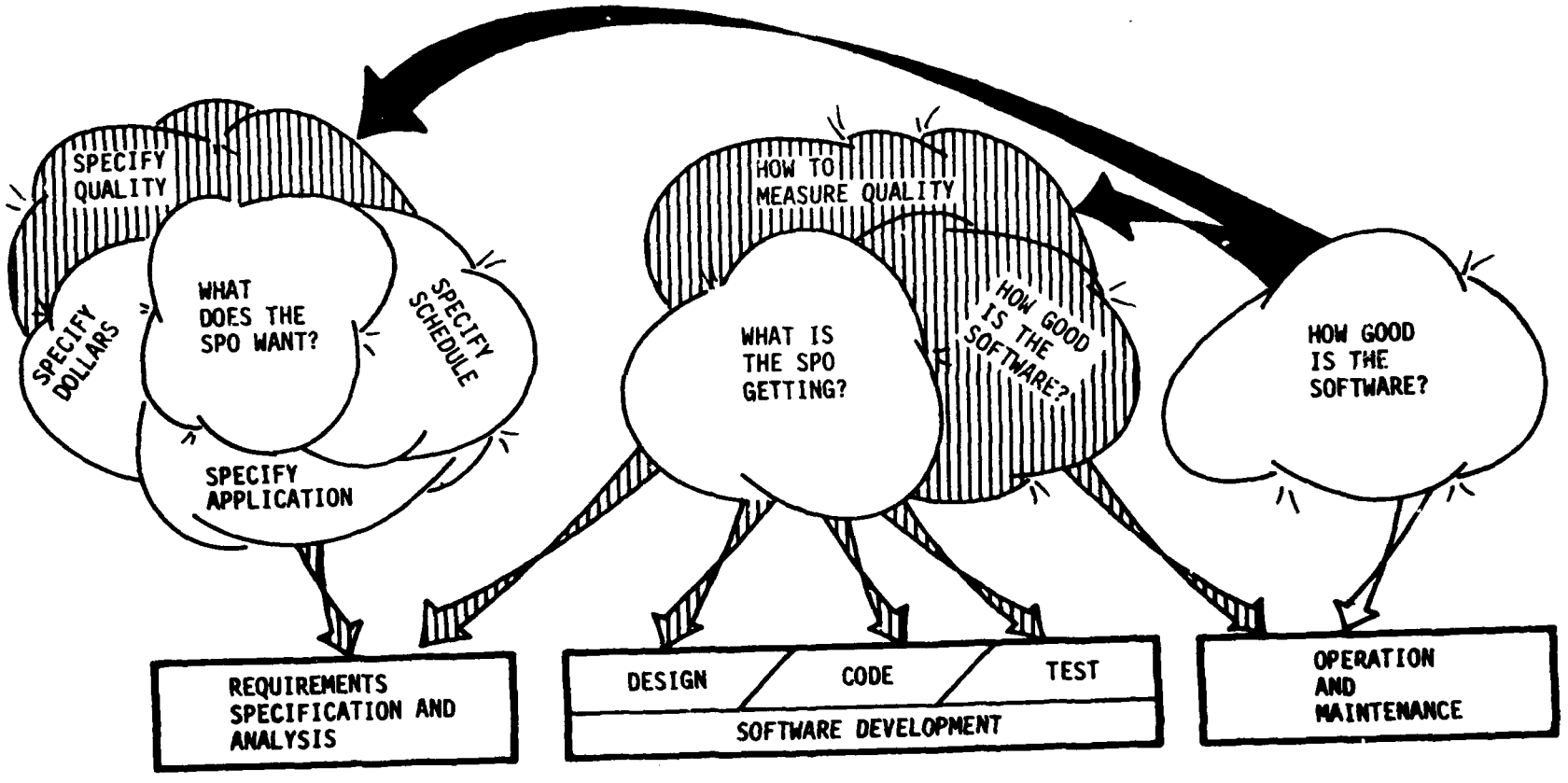


Figure 1.2-1 Specifying and Measuring Quality Software

In taking this approach, we established a comprehensive framework which facilitates the incorporation of future efforts and refinements to the metrics and their correlation to the quality factors. Also, recommendations are made on how the metrics should be collected.

The results of this task provide the SPO with a methodology for specifying the quality he wants in the software and the procedures for determining if he is realizing the level of quality that was specified. By achieving this goal, the SPO will have objective insight into the software quality throughout the acquisition process.

1.3 ACKNOWLEDGMENT OF PREVIOUS WORK

In establishing the framework for this study of factors in software quality, we are attempting to incorporate the work that others have done previously in this area. An extensive literature search was conducted. The references are listed following the Appendices and the major references are abstracted in the bibliography.

We used other RADC sponsored efforts, particularly those in the area of reliability and maintainability, as input to this task. The planned approach was to concentrate in those areas where little work has been done.

1.4 CONTRIBUTION TO STATE OF KNOWLEDGE

This study has been directed at expanding upon the current state of knowledge about software quality. The following aspects of our approach are identified as expansions to the work to date:

- Provide a global view of software quality - most previous efforts have evaluated subsets.
- Provide a formal methodology for the validation of metrics.
- Relate the quality factors to Air Force applications.
- Relate the quality factors to the life cycle phases.

- Define metrics which are programming language independent.
- Identify metrics which can be applied early in the development phase (during requirements analysis and design).
- Attempt to choose criteria that are as independent and unambiguous as possible.
- Attempt to quantify the correlation of subjective criteria to the quality factors.
- Identify automated metric data collection tools.
- Provide a framework for factors in software quality that can be used in future research efforts.

1.5 CONCLUSIONS OF THE STUDY

The effort represents a conceptual study investigating the factors of software quality. Our intent was to build upon the significant contributions of other efforts in recent years related to understanding software quality. The main thrusts of this study were the formulation of an SPO-oriented concept of factors in software quality and the establishment and application of metrics oriented toward the early phases of development. The measures are indicators of the progress toward the desired quality. They also give an early indication of the quality factors that are not realized in testing or during initial operation but have a large cost impact later in the life of a product, e.g., portability, reusability, or interoperability.

The complete procedure of establishing a framework for factors in software quality, defining the factors, relating them to Air Force applications and the life-cycle phases, establishing criteria, defining them, using them to identify the relationships and tradeoffs between factors, defining metrics, establishing their relationship to the quality factors, and validating the relationship was an iterative process. It has been described in this report in a sequential manner only for clarity and simplicity. It will continue to evolve as more experience is gained through the application of metrics to more software developments.

The framework established is flexible and expandable. It provides a complete view of software quality. It provides a mechanism for specifying and measuring the quality of a software product. The following benefits can be realized from this conceptualization of factors in software quality:

- it is a simple, comprehensive tool for an acquisition manager to use – guidelines for its use are provided in the form of a handbook. (Volume III)
- it provides the acquisition manager with a life-cycle view of his software product, forcing consideration of such factors as maintainability and portability in the system specification phase.
- it provides a mechanism for performing high-level tradeoff studies early in the life cycle (requirements analysis, performance requirements analysis, and preliminary design) to help in determining the product's required capabilities and performance characteristics.
- as the software development process technology advances and new development techniques are introduced, the metrics can easily and logically be modified or added.

The set of metrics established provides a comprehensive coverage of the characteristics of a software product. As they exist, they represent an excellent guideline for testers, quality assurance personnel, and independent verification and validation efforts. They also incorporate an extensive composite of a number of texts on good programming practices and style.

The specific results of the validation phase of the study allow the conclusion that software metrics are a viable concept. The regression analysis showed significant correlation for some metrics with related quality factors. Quantitative metrics can be applied to intermediate products of the software development which exist as early as the requirement analysis. As more disciplined, software engineering approaches are taken toward the development of software, the more applicable quantitative metrics become.

The establishment of generalized precise normalization functions was beyond the scope of the study. The limiting factors were that the sample of modules and systems was not large enough, general enough, nor had the two systems, which were used, been through all of the quality factors related activities (e.g., moved to another environment, linked to another system, etc.). The sample was representative of two large-scale developments so the experience of applying the metrics contributed considerable knowledge to the software quality technology. One other limiting factor was that the measures were biased high because the metrics were applied after the two systems had been delivered. So, even though the metrics were applied to software products delivered during the development they had been updated to reflect all of the changes and fixes made to the system as a result of testing and operational experience. A definite recommendation of this study then is to apply the metrics during the actual development of a software system to further validate their relationship to the resulting quality.

In deriving the set of metrics, the number of metrics became a significant consideration. The concept of applying the same metric successively during the development phases helped contain the problem of an unwieldy number of metrics. The fact that many of the metrics can be collected automatically assists in making the present set more manageable.

A large number of existing software support tools were identified that provide metric data collection capabilities. Significantly, several tools were identified, and some applied, which automate the collection of metrics in the requirements analysis and design phases of the development. Several other tools can be developed. Because many tools do exist that provide a subset of the overall capabilities of data collection required, an integrated approach must be developed to effectively collect metric data in any software development environment.

Some very practical, beneficial results from the application of the metrics in their current form have been identified. When the metrics are applied to the set of software products available at various times during the development, they can be used as indicators. Low measurements identify modules or characteristics which should be investigated and the scores justified. The methodology for regression analysis described can be used in conjunction with this metric indicator concept. The analysis provides an indication of what specific software characteristics vary in a particular environment relative to variations in software quality, i.e., which characteristics vary significantly and cause variation in the software quality.

This information is beneficial to software developers in writing their design and programming standards and conventions. It is also beneficial to QA personnel in identifying areas or modules requiring attention during development and concentrated testing.

An SPO can use the quantitative nature of the metrics and the framework of the software quality factors to specify the required level of software quality quantitatively. By specifying the software quality in terms of the metrics, the SPO is specifying the desired characteristics of the software. The characteristics are essentially independent of method or philosoply of software development so there are no unjustified restrictions placed on the software developer.

The software quality metrics represent the introduction of a more disciplined engineering approach to software quality assurance. They provide a quantitative tool for the inspection and evaluation of software products during the development phase.

1.6 FURTHER RESEARCH

Several areas for further research were identified during this effort.

The exercise of determining the set of metrics revealed several areas requiring further investigation. Within the transition phase, the two quality factors, reusability and interoperability, are relatively untouched in the literature. Little research has been conducted to determine what constitutes reusability and interoperability or what software attributes provide these qualities. It is felt that further research in these areas could have potentially high life-cycle cost benefits.

A second area where we feel further research would be beneficial is in measuring various aspects of efficiency. Because many of the attributes of efficiency have a negative effect on all other quality factors, it is an important consideration of the software quality concept. Most current measures of efficiency are dynamic measures requiring execution of the code. In deriving some static measures we realized that an integrated set of both dynamic and static measures are necessary to judge the degree of efficiency. Further work is required to develop this type of measure.

Further research, application, and experience are required to formalize the normalization functions. This report has stressed the methodology of deriving and validating the normalization functions to encourage the application of these techniques to other software developments. Use on future developments will add to the data base for the establishment of generalized normalization functions, as well as provide indication to the SPO and software developer of their progression toward a high quality product. It will also contribute to the error data collection technology and experience.

As previously mentioned, the metrics should be applied during a software development to obtain more realistic measures. It is also recommended that the metrics be applied to specific projects involving (1) software conversions from one environment to another to validate the metrics related to portability, (2) efforts linking two systems to validate the interoperability metrics, and (3) efforts upgrading a system to validate the reusability metrics. These efforts would not only provide a chance

for validation of the particular metrics but also give considerable insight into additional metrics in these high-payoff, late-life-cycle-impact quality factors.

SECTION 2

DETERMINATION OF QUALITY FACTORS

2.1 DEFINITION OF TERMS

To be consistent in our determination of factors, criteria, and metrics, we first established a set of working definitions. This was done in order to provide a framework from which to more objectively judge candidate quality factors. The working definitions are as follows:

- 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 were used to determine the prime set of 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 2.1-1 illustrates the relationship between a factor and the cost to achieve different levels of that quality factor. As an example, we will assume the curve describes the cost to 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

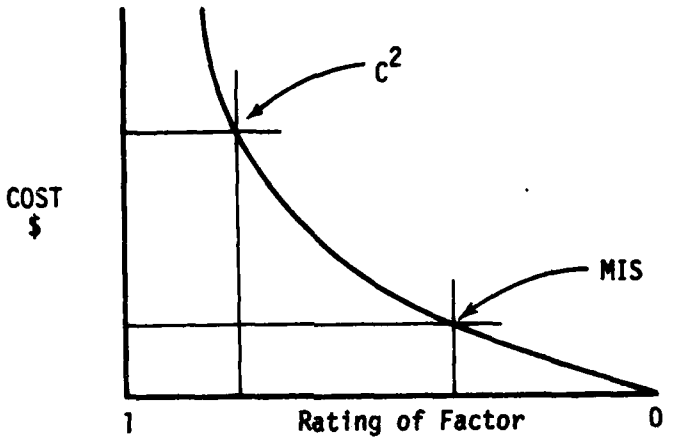


Figure 2.1-1 Relationship of Software Quality to Cost

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 applications. So, while the C^2 final product may have a higher degree of reliability according to our measures, it is no more acceptable to its user than the MIS system with its lower reliability is to its user. This relationship is further illustrated in Section 3 where the quality factors are related to specific Air Force applications.

- Criteria: attributes of the software or software production process by which the factors can be judged and defined. The following rules were applied to the determination of 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.
- Metrics: measures of the criteria or subcriteria 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. Metrics will be discussed further in Section 6.

2.2 IDENTIFICATION OF QUALITY FACTORS IN THE LITERATURE

A literature search was conducted to assemble all current definitions and to identify any applicable discussions with respect to software quality factors. Table 2.2-1 summarizes the list of terms extracted from the literature and represents the baseline of potential or candidate quality factors referenced in this study.

This list of approximately 55 terms was used as the starting point for determining the prime set of factors. The next task was to apply the definitions given in Section 2.1 to the list of candidate factors. The intent of this exercise was to put into place a standard by which to judge terms with regard to consistency, redundancy, suitability, etc. The results of applying the definitions to the candidate terms is discussed in further detail below, where the rationale for terms such as understandability, modularity, and complexity is explained.

In Table 2.2-2 we provide a brief cross-reference of definitions and authors quoted. The total set of definitions analyzed in this report appears in Appendix A where work by various researchers in the software community are quoted or paraphased.

2.3 THE PROCESS OF GROUPING CANDIDATE FACTORS

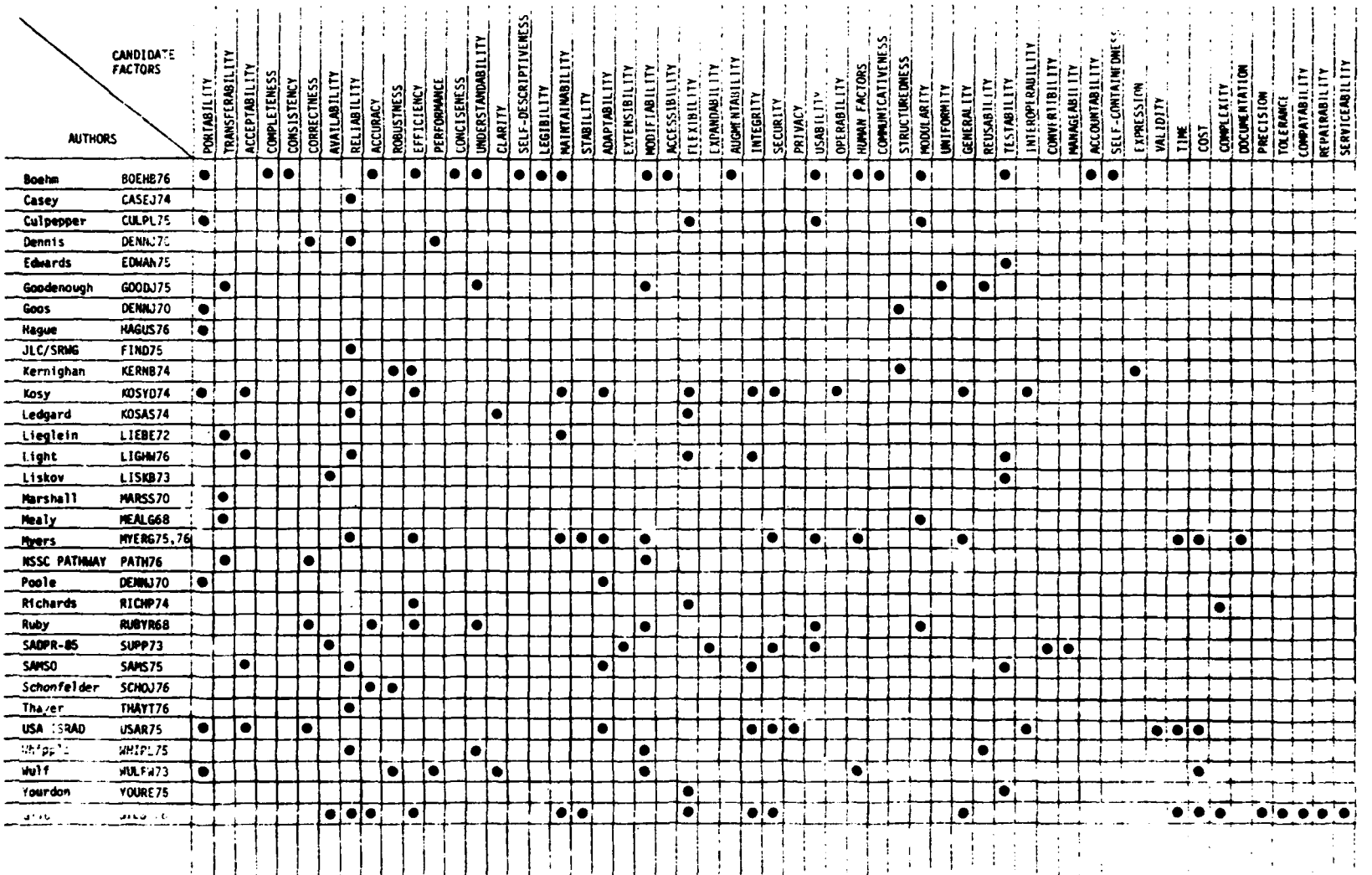
The list of potential factors established in Table 2.2-1 was known to contain obvious redundancy and some terms which do not comply with all of the rules identified for the prime set of factors. It was also felt that the list was far too long to represent a manageable set of factors. For this reason, some guidelines were generated to aid in grouping the factors into a smaller, concise number of entries which still cover the comprehensive set of software quality factor characteristics desired. The guidelines used were:

- User-oriented terms are potential factors; software-oriented terms are potential criteria.
- Synonyms that are identified are grouped together.

Table 2.2-1 Candidate Software Quality Factors Extracted from the Literature

PORTABILITY	AUGMENTABILITY
TRANSFERABILITY	INTEGRITY
ACCEPTABILITY	SECURITY
COMPLETENESS	PRIVACY
CONSISTENCY	USABILITY
CORRECTNESS	OPERABILITY
AVAILABILITY	HUMAN FACTORS
RELIABILITY	COMMUNICATIVENESS
ACCURACY	STRUCTUREDNESS
ROBUSTNESS	MODULARITY
EFFICIENCY	UNIFORMITY
PERFORMANCE	GENERALITY
CONCISENESS	REUSABILITY
UNDERSTANDABILITY	TESTABILITY
SELF-DESCRIPTIVENESS	INTEROPERABILITY
CLARITY	CONVERTIBILITY
LEGIBILITY	MANAGEABILITY
MAINTAINABILITY	COST
STABILITY	ACCOUNTABILITY
ADAPTABILITY	SELF-CONTAINEDNESS
EXTENSIBILITY	EXPRESSION
MODIFIABILITY	VALIDITY
ACCESSIBILITY	TIME
FLEXIBILITY	COMPLEXITY
EXPANDABILITY	
PRECISION	DOCUMENTATION
TOLERANCE	REPAIRABILITY
COMPATABILITY	SERVICEABILITY

Table 2.2-2 Sources for Software Quality Factors



- Logically similar terms are grouped together.
- Required a manageable number of groups - the prime set of factors should be small in number to maintain a simple framework in which the SPO can work most effectively.

A central issue in the grouping process was to identify only user-oriented terms as potential factors. This has the advantage of defining for the SPO a set of user-oriented terms for specifying the relative amount of quality desired in the product. It also enabled us to significantly reduce the number of potential factors. In defining software-oriented terms as criteria, then, we also could establish more realistic standards by which to judge the final software product which will be, in turn, easier to measure.

2.4 RESULTS AND RATIONALE AFTER GROUPING QUALITY FACTORS

Table 2.4-1 provides the results of the grouping process. The underlined terms were chosen as the group names. They were chosen because they were the most descriptive or, if a hierarchical relationship existed in the group, the higher member was chosen.

Three groupings were determined not to be candidates for a quality factor: Understandability, Complexity, and Modularity. Complexity and modularity are software-oriented rather than user-oriented terms. The user is interested in such things as how fast the program runs (efficiency) and how easy it is to maintain (maintainability), not how modular it is. Modularity is not an end item quality or performance characteristic. Since it is software oriented, it contributes to several of the candidate quality factors, and is therefore a candidate for a criterion. Similarly, complexity is a candidate for a criterion (simplicity will be used to connote a positive attribute).

Understandability was initially identified as a quality factor. Upon further analysis (relating factors to Air Force applications and to life-cycle phases), we decided that it was not quantifiable. Measuring how well software is understood is extremely difficult and associating a cost would be even more difficult. The reasons for wanting to understand a program really are related to using, maintaining, or changing the program. Thus, the quantifiable attributes of understandability will be used as criteria of other factors.

Table 2.4-1 Grouping of Software Quality Factors to Achieve Unambiguous Set

CORRECTNESS	UNDERSTANDABILITY	PORTABILITY
Acceptability	Clarity	Transferability
Completeness	Legibility	Compatability
Consistency	Self-Descriptiveness	
Expression		
Validity		
Performance		
RELIABILITY	MAINTAINABILITY	REUSABILITY
Availability	Stability	Generality
Accuracy	Manageability	Utility
Robustness	Conciseness	
Precision	Repairability	
Tolerance	Serviceability	
EFFICIENCY	FLEXIBILITY	INTEROPERABILITY
	Adaptability	
	Extensibility	
	Accessibility	
	Expandability	
	Augmentability	
	Modifiability	
INTEGRITY		COMPLEXITY
Security		
Privacy		
USABILITY	TESTABILITY	MODULARITY
Operability	Accountability	Structuredness
Human Factors		Uniformity
Communicativeness		Self-Containedness
Convertibility		
DOCUMENTATION	COST	TIME

Integrity, security, and privacy were grouped together. Integrity is interpreted as the ability of the software to protect against unauthroized access - software integrity. Another common interpretation, data base integrity, (the ability of the software to maintain an accurate data base in a multiaccess environment) falls under the major category of reliability. Privacy is the ability to control the use of data. An authorized person may access data and then use it in an unauthorized manner. To date, little software has been developed for providing the capability to control usage of data. In this respect, privacy is outside the scope of this study. We will maintain it as a part of the quality factor called integrity for future expansion.

Two other groupings, cost and time, provide the baseline for evaluating the factors. It costs more and takes more time to develop a more reliable system. A system that is not flexible costs more and takes longer to change. Cost and time, therefore, were not considered candidates for quality factors, but form the basis for correlating metrics with the various levels of quality. Documentation is one of the vehicles for providing the various qualities. Specific documents enhance the maintainability, others the testability, and so on. Documentation was not considered a quality factor but a product of the software development process which can be evaluated to obtain a measure of quality.

SECTION 3

DEFINITIONS OF QUALITY FACTORS

3.1 CONCEPTUALIZATION OF FACTORS IN SOFTWARE QUALITY

Through evaluation and analysis of the groupings of factors, three different orientations that one could take in looking at a delivered software product were recognized. These three orientations took on added significance because of their relevance to an SPO. The SPO's interaction with a delivered software product can be described in terms of only three distinct activities as follows:

Software Product Activities

- Product Operation
- Product Revision
- Product Transition

This scheme was adopted as the framework about which we would further evolve our software quality research. Figure 3.1-1 illustrates the concept derived. The questions in parentheses provide the relevancy or interpretation of the factors to an SPO.

To date, the major emphasis has always been on initial product operation. Specifications and testing only stress factors such as the correctness or reliability. There is, however, a growing recognition of the longer-term implications in our software development, such as the flexibility or maintainability of the software product. Analyses of life-cycle costs have shown that the costs of maintenance and redesign exceed the cost of initial development [LIEBE72] and that the cost of fixing errors after a system is operational is up to 30 times greater than if the error was caught during system testing. Several software developers state as policy a rule of thumb that if a modification requires X% of the lines of code to be revised, a total redesign is undertaken; X varies between 20 and 50 percent. All of these facts point to the conclusion that our software systems are not designed and developed with the required emphasis on maintainability, flexibility, portability, etc.

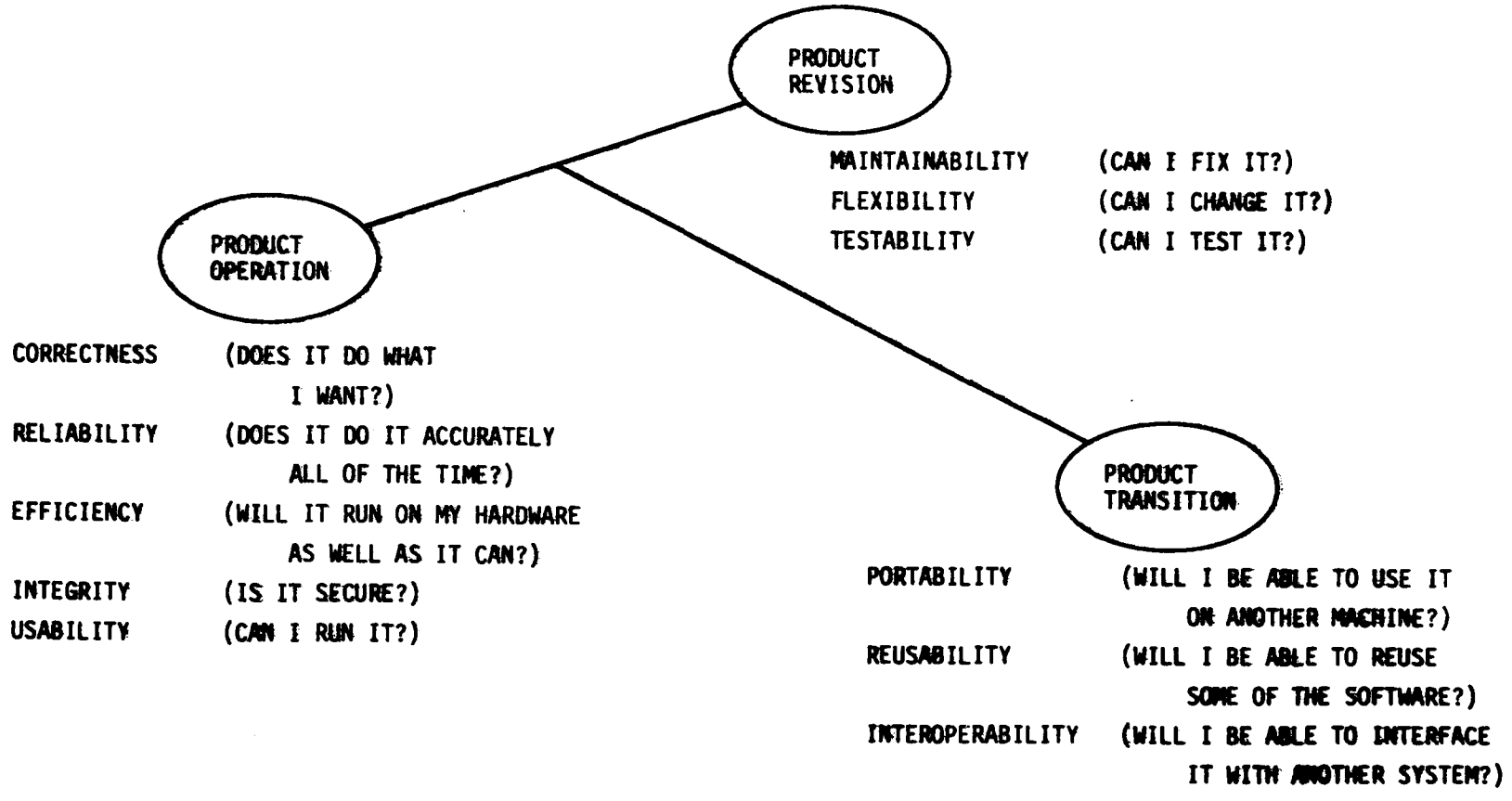


Figure 3.1-1 Allocation of Software Quality Factors to Product Activity

Significant impacts to the total cost of a system during the life of the system can be realized because of the following reasons:

- maintenance and operations costs are very high
- major program modifications are required
- there is a change of mission
- a change occurs in the hardware
- there is a change of users, either a new Air Force user or new contractor

An occurrence of any of the above events usually requires redesign, recoding, and retesting. The size of the impact of one of these events on total system cost is a function of the quality factors associated, in particular, with product revision and product transition. In order to minimize the impact, these factors must be taken into account in the initial program development.

The Air Force Systems Command, in order to provide more of a focus on these problems, has initiated efforts to improve their total life-cycle management techniques [AIRF76]. The Naval Sea Systems Command has initiated the PATHWAY Program, in which they recognize the importance of portability and reusability in a software product [PATH76].

This conceptualization of factors in software quality provides a mechanism for the SPO to quantify these concerns for the longer life-cycle implications of the software product. 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 system 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 from time to time, reusability is of prime importance in those modules which implement the major functions of the system. With the advent

of more networks and communication capabilities, more systems are being required to interface with others and the concept of interoperability is extremely important. All of these considerations can be accommodated in the framework derived.

The formal definitions for each of the quality factors are described in Table 3.1-1.

3.2 RELATIONSHIP OF FACTORS TO AIR FORCE APPLICATIONS

We conducted further evaluation of the framework and the quality factors by examining their applicability to Air Force applications. To make this evaluation, a categorization of Air Force applications was derived. This categorization is shown in Table 3.2-1. Several references [AIRF76, THEE75, SACI76, SACKH67] were utilized in deriving this scheme. We found, however, that the factors vary considerably within the categories as well as between categories, depending on the specific application. So we chose specific systems representative of the categories and had several people familiar with Air Force missions identify the importance of the factors to the specific software product. The ratings shown in Table 3.2-2 are the means of the individual ratings and were calculated as follows:

V - Very High - 6 points
H - High - 4 points
M - Medium - 3 points
L - Low - 1 point

$$\text{Average rating} = \frac{\sum_{n}^{n} r_i}{n}$$

where r_i = individual ratings
n = total number of people rating

Table 3.1-1 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.

Table 3.2-1 Categorization of Software in Air Force Systems

CATEGORIES	TYPICAL CHARACTERISTICS	TYPES OF AF SYSTEMS	SPECIFIC EXAMPLES
DEVELOPMENT/TEST BED	Micro to maxi/timeshare to batch.	R&D Lab Software House	RADC H6000 National Software Works (NSW)
STUDY/SIMULATION	Maxi Batch or real time	Flight simulator Mission Planning Training	B52, B1 Electro-Optical Viewing System. Experimental Radar Prediction Device B52, KC135 Weapon System Trainer
MANAGEMENT INFORMATION SYSTEM	Micro to maxi On-line/Real time or batch	Personnel Finance Logistics Inventory Control	Installation Data Processing Ctr. MACIMS - Military Airlift Command Information Management System
COMMUNICATIONS	Micro or mini Real Time Networks - Front End Processors	Communication Networks	SAC Automated Total Information Network (SATIN IV) Joint Tactical Information Distribution System (JTIDS) World Wide Military Command & Control System (WWMCCS)
STRATEGIC/TACTICAL COMMAND AND CONTROL	Maxi Real time	Command and Control	National Military Command System (NMCS) Advanced Airborne Command Post
SENSOR DATA PROCESSING/ INTELLIGENCE	Maxi or mini Real time	Space Defense Air Defense Remotely Piloted Vehicles	Tactical Information Processing and Interpretation (TIPI) Ballistic Missile Early Warning System (BMEWS) Cobra Dane Phased-Array Radar Pave Paws
INDICATIONS AND WARNING	Maxi with front end processors Real time	Radar Early Warning Air Traffic Control	Airborne Warning & Control System (AWACS) North American Air Defense (NORAD) Over the Horizon Radar (OTH) Backup Interceptor Control (BUIC) SAGE Traffic Control & Landing System (TRACALS)

Table 3.2-1 Categorization of Software in Air Force Systems (continued)

CATEGORIES	TYPICAL CHARACTERISTICS	TYPES OF AF SYSTEMS	SPECIFIC EXAMPLES
UNMANNED SPACECRAFT/ MISSILES	Mini on board processor Maxi ground stations Networks Real time	Satellites Missiles Anti-Missiles Weather	Defense Meteorological Satellite System (DMSP) Defense Support Program (DSP) Defense Satellite Command System (DSCS) Global Positioning System (GPS) Minuteman ICBM Advanced Missile (MX)
MANNED SPACECRAFT/ AIRBORNE AVIONICS	Mini on board processor Maxi ground stations Real time	Satellites Flight Control Weapon Systems Radar Electronic Countermeasures	Space Transportation System (STS SHUTTLE) FB-111 SAC Medium Bomber F-18, F-16, F-15 Fighters

Table 3.2-2 Importance of Software Quality Factors to Specific Air Force Applications

EXAMPLE SYSTEMS	EXPLANATION	FACTORS: PRODUCT OPERATION — CORRECTNESS	RELIABILITY	EFFICIENCY	INTEGRITY	USABILITY	PRODUCT REVISION — MAINTAINABILITY	FLEXIBILITY	TESTABILITY	PRODUCT TRANSITION — PORTABILITY	REUSABILITY	INTEROPERABILITY
RADC H6000/MSW	Development/Test Bed/R&D Lab type System	L-M	M	L	L-M	H	L-H	H	M	H	H	L
Graduate Pilot Flight Simulator	Pilot Training	H	H	H	L	H-V	M	M-H	M	L	L	L
Personnel System at AF Base	Personnel status, assignments, promotions, etc.	H	M	L-M	H	H	M-H	L	M	L-M	M	L-M
MACIMS - Military Airlift Command Information Management System	Allocation of Aircraft	H	H	H	M-H	H	H	M	M	M	M	H
SATIN IV-SAC Total Information Network	World Wide Communication System	H	H-V	H-V	V	H-V	H	M	H	L	L	H
E-4 Advanced Airborne Command Post	Survivable C^2 Airborne Operation	H-V	V	H-V	H-V	H-V	H	H	H	L	L	H
Over the Horizon Radar	NORAD surveillance and warning system. Detect targets at all altitudes.	H-V	V	H	H	H	H	M-H	H	L	L	H
TIPI - Tactical Info Processing & Interpretation System	Intelligence data processing for the tactical commander. Air transportable mobile shelters.	H	V	H	V	H	H	H	H	M	M	H
TRACALS - Traffic Control and Landing Systems	Air traffic control including Terminal Navigation Aids, Landing Systems, Air Traffic Control Simulators	V	V	H	M-H	V	H	M	H	L-M	L	M
MINUTEMAN	ICBM Offensive Weapon System	V	V	H-V	H	L-M	H	M	V	L	L	L
DMSP - Defense Meteorological Satellite Program	Weather, Mapping Applications	H	H	M-H	M	H	H	M	H	L	L	L
DSP- Defense Support Program	Defense Oriented Satellites	V	V	H	H-V	H	H-V	H	V	L	L-M	M
STS - Space Transportation System (SHUTTLE)	Manned Spacecraft - High Visibility Program	V	V	H	H	V	H-V	H	V	L	L	L-M
FB-III Medium Bomber Airborne Avionics	Includes navigation and weapons control, inertial navigation, terrain following radar, etc.	V	V	H-V	H	H	H	H	H-V	L	L	L
AVERAGE RATING OF FACTORS		H-V	H-V	H	H	H	H	M	H	L	L	L-M

LEGEND

Level of importance of quality factors:

V - Very High

H - High

M - Medium

L - Low

Rating in matrix was assigned according to following:

Average Rating	Rating in Matrix
1≤ - <2	L
2≤ - <2.5	L-M
2.5≤ - ≤3.5	M
3.5	M-H
3.5< - ≤4.5	H
4.5< - ≤5	H-V
5< - 6	V

The representative set of missions is shown against the set of software quality factors. The missions range from a software test bed laboratory facility to a man-rated space mission (STS). The ratings illustrate the relative qualities that are evidenced by the specified goals of the systems.

As shown, the factor of reliability is absolutely critical for the success and safety required for the spacecraft and early warning applications. On the other hand, while reliability is always a desirable quality, the degree necessary for success in the management information system and test bed applications decreases relative to the other missions. The amount of efficiency required is determined not only by the application itself but also by the computing resources available and loading expected during a given mission. On-board processors, because of size limitations, usually have to realize a high degree of efficiency. A communication network, handling large volumes of data at a high rate, has to be efficient in its data switching and handling capabilities. As another example, a large space mission application dedicated to a particular hardware as well as software environment may not consider portability an essential factor, whereas, in the test bed facility, the ability to transport software to and from other configurations could well be one of the most important considerations in the entire system.

In filling out Table 3.2-2, we found that very few highs (H) or very highs (V) were given to quality factors in the product revision or product transition categories. This illustrates the lack of attention given these areas in past software development. The average ratings were calculated to point out this fact. The quality factors associated with product operation all received high or very high ratings. These were expected and are probably justified by the current practices and procedures in use today and utilized in specifying software development projects. The quality factors associated with product revision rated somewhat lower and, except for a few high-to-very high and very high ratings in maintainability and testability, would have been even lower. The quality factors associated with product transition all rated medium to low. This, too, points to the lack of attention given to the problems associated with software conversions.

There may be more basic system characteristics that have more effect on the factors than the categorization scheme. A partial list with the factors that are extremely important is provided below:

- if human lives are affected (Reliability, Correctness, Testability)
- very high system development cost (Reliability, Flexibility)
- long life cycle (Maintainability, Portability, Flexibility)
- real time application (Efficiency)
- on-board application (Efficiency, Reliability)
- processes classified information (Integrity)
- interrelated systems (Interoperability).

3.3 RELATIONSHIP OF FACTORS TO LIFE-CYCLE PHASES

Further evaluation of the framework and quality factors was needed to insure coverage of the entire life cycle of a software product and to determine if early indication of the quality was possible. Table 3.3-1 identifies where the factors should be measured (Δ) and where the greatest impact due to poor quality can be expected (X).

For instance, the reliability of a system is immediately in jeopardy if a contractor does not understand the mission requirements or does not detect this

Table 3.3-1 The Impact of not Specifying or Measuring Software Quality Factors

	DEVELOPMENT			EVALUATION	OPERATION		
LIFE-CYCLE PHASES / FACTORS	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

fact during requirements analysis or system design phases. The system may then proceed through code and debug and even into test and operation before the impact of this failure is recognized. The cost and effort involved in reevaluation, redesign, recoding, and retesting is significantly greater than if realized earlier. Maintainable and efficient code can be achieved only by designing and coding these properties into the software.

From the table we see that all of the factors can be measured during the design phase. Most of the research in metrics to date has been in the code and debug area. Determination of quality is more subjective in nature at the design level than during the code and debug phase, where metrics can be applied to code. This identifies where emphasis for research should be placed to take advantage of the cost savings of early detection of poor quality. As an example of a more objective measure in the design phase, a traceability matrix, which relates performance and design requirements to the original system requirements specification, can be effectively utilized to evaluate the quality of work performed by the contractor during requirements analysis.

SECTION 4
DEFINITION OF CRITERIA

4.1 DEFINING FACTORS WITH CRITERIA

The establishment of criteria for each factor has a four fold purpose. First, the set of criteria for each factor further defines the factor. Second, criteria which affect more thar. one factor help describe the relationships between factors. Third, the criteria allow a one-to-one relationship to be established between metrics and criteria. Lastly, the criteria further establish the working hierarchical nature of the framework for factors in software quality.

As the software development technology progresses, other metrics, criteria or even factors may be identified as relevant to the needs of an SPO. The framework being established allows for and facilitates this kind of expansion or refinement by its hierarchical nature.

The set of criteria for each quality factor are shown in Figure 4.1-1. The factors are identified in ellipses and the criteria are identified in rectangles. These criteria were derived utilizing the software-related terms from Table 2.4-1, examining the definition of each factor and expanding it into independent attributes, and by identifying criteria with which we can potentially associate objective measures.

For example, the attributes or standards for reliability are error tolerance, consistency, accuracy, and simplicity. Integrity connotes protection which implies two forms of protection: access control and access audit.

The definitions of the criteria are provided in Table 4.1-1.

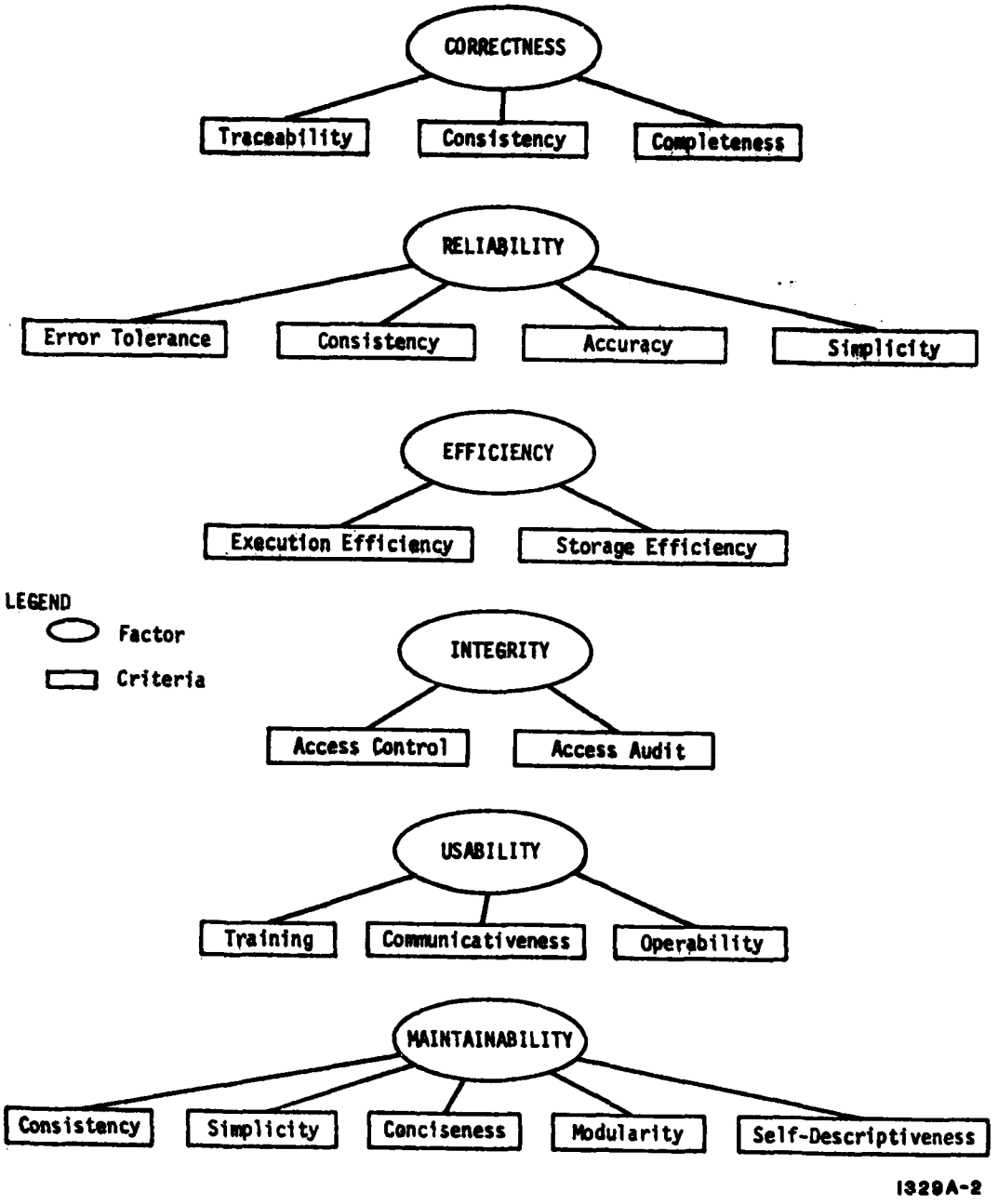


Figure 4.1-1 Relationship of Criteria to Software Quality Factors

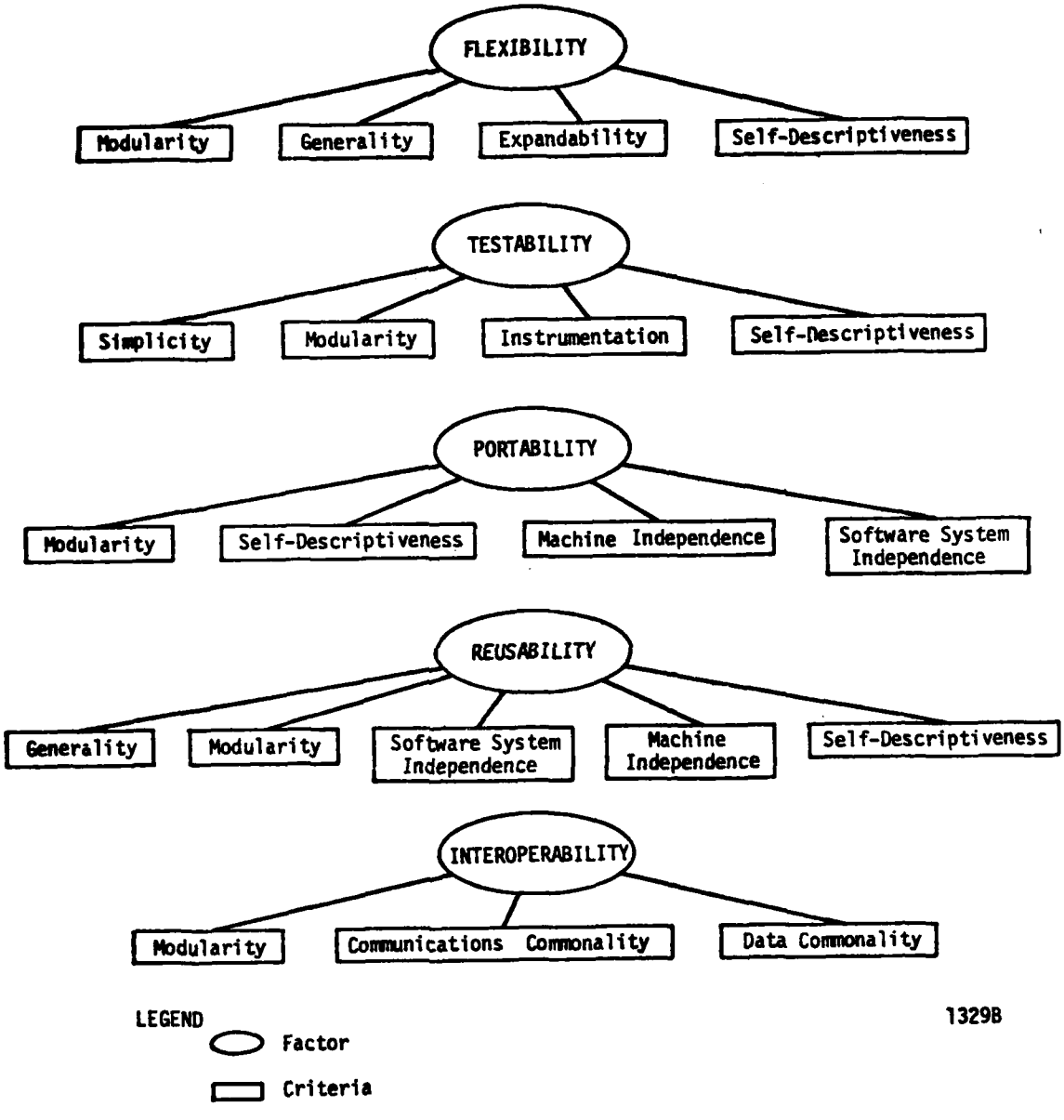


Figure 4.1-1 Relationship of Criteria to Software Quality Factors (continued)

Table 4.1-1 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 4.1-1 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

4.2 RELATIONSHIP BETWEEN FACTORS

The conceptualization of the factors in software quality implies some relationships between the factors. Those grouped under Product Operation, Product Revision, and Product Transition are related simply by association with those aspects of a software product's life cycle. These relationships are very high level, user-oriented interactions.

The criteria, especially those common to more than one factor, provide a much more detailed understanding of the factors and their relationships to one another. Table 4.2-1 depicts when the criteria should be measured (Δ) and when impact of poor quality will be realized (X) during the life-cycle phases. This table is an expansion of Figure 3.3-1, the impact of not specifying or measuring software quality factors.

The effect of each criteria on each factor was evaluated and the results are displayed in Table 4.2-2. If the existence of the attributes characterized by the criterion positively impact a given factor, a (o) was placed in the matrix. If the existence of the attributes characterized by the criterion negatively impacts the factor, a (●) was placed in the matrix. If there was no relationship between the criterion and factor, if the relationship was not clear, or if it was highly dependent on the application, a blank was placed in the matrix. These criterion/factor relationships are the basis for the factor-to-factor relationships. If all of the criteria of one factor have positive impacts on another factor, then the relationship of those two factors is very positive. Conversely, if all of the criteria of one factor have negative impact on another factor, these two factors display a negative (tradeoff) relationship.

As an example, consider the factors of portability and efficiency. Two criteria of efficiency, execution efficiency and storage efficiency, have negative impacts on portability. Conversely, two criteria of portability, software

Table 4.2-1 Impact of not Applying Criteria in Specifying Software Quality

LIFE CYCLE PHASES / CRITERIA	DEVELOPMENT			EVALUATION	OPERATION		
	REQMTS ANALYSIS	DESIGN	CODE & DEBUG	SYSTEM TEST	OPERATION	MAINTENANCE	TRANSITION
Traceability	△	△	△	X	X	X	
Completeness	△	△	△	X	X		
Consistency		△	△	X	X	X	
Accuracy	△	△	△	X	X		
Error Tolerance	△	△	△	X	X		
Simplicity		△	△	X	X	X	X
Modularity		△	△	X	X	X	X
Generality		△	△			X	X
Expandability		△	△			X	X
Instrumentation		△	△	X	X	X	X
Self-Descriptiveness			△	X		X	X
Execution Efficiency		△	△		X		
Storage Efficiency		△	△		X		
Access Control	△	△	△		X		
Access Audit	△	△	△		X		
Operability	△	△	△	X	X		
Training		△	△		X		
Communicativeness		△	△	X		X	
Software System Independence		△	△			X	X
Machine Independence		△	△				X
Communications Commonality	△	△	△				X
Data Commonality	△	△	△				X
Conciseness			△			X	X

LEGEND: △ Where criteria should be measured X Where impact of poor quality is realized

Table 4.2-2 Effect of Criteria on Software Quality Factors

CRITERIA \ QUALITY FACTORS	CORRECTNESS	RELIABILITY	EFFICIENCY	INTEGRITY	USABILITY	MAINTAINABILITY	TESTABILITY	FLEXIBILITY	PORTABILITY	REUSABILITY	INTEROPERABILITY
TRACEABILITY	○					○	○	○		○	
COMPLETENESS	○	○			○						
CONSISTENCY	○	○				○	○	○		○	
ACCURACY		○	●		○						
ERROR TOLERANCE	○	○	●		○						
SIMPLICITY	○	○	○			○	○	○	○	○	
MODULARITY			●			○	○	○	○	○	○
GENERALITY		●	●	●				○		○	○
EXPANDABILITY			●					○		○	
INSTRUMENTATION			●		○	○	○				
SELF-DESCRIPTIVENESS			●			○	○	○	○	○	
EXECUTION EFFICIENCY			○						●		
STORAGE EFFICIENCY			○				●		●		
ACCESS CONTROL			●	○	○			●			●
OPERABILITY			●		○					○	
TRAINING					○					○	
COMMUNICATIVENESS			●		○	○	○	○		○	
SOFTWARE SYSTEM INDEPENDENCE			●					○	○	○	○
MACHINE INDEPENDENCE			●					○	○	○	○
COMMUNICATIONS COMMONALITY											○
DATA COMMONALITY				●						○	○
CONCISENESS	○		○			○	○				
ACCESS AUDIT			●	○							

LEGEND - Attributes associated with criteria have:

● Negative effect on quality factor ○ Positive effect on quality factor

system independence and machine independence, negatively impact efficiency. The relationship between efficiency and portability is quite obvious in that when a high degree of portability exists, you can expect a low degree of efficiency. Where some criteria of a factor impact another factor positively and some negatively, further analysis is required to determine the net effect of the impact.

Table 4.2-3 displays the results of the criteria versus factors analysis at the factor versus factor level. In this table, the (o) indicates that with the existence of a high degree of one factor you would expect a high degree of another factor. A (●) indicates that with a high degree of one factor you would expect a low degree of the other factor.

Looking at portability versus efficiency situation, we find a (●) entered in the matrix shown in Table 4.2-3. This indicates the tradeoff situation already discussed. This table then describes the general relationships between factors. Specific cases must be analyzed along these lines to determine the specific tradeoffs. A discussion of the tradeoffs generally found follows:

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.

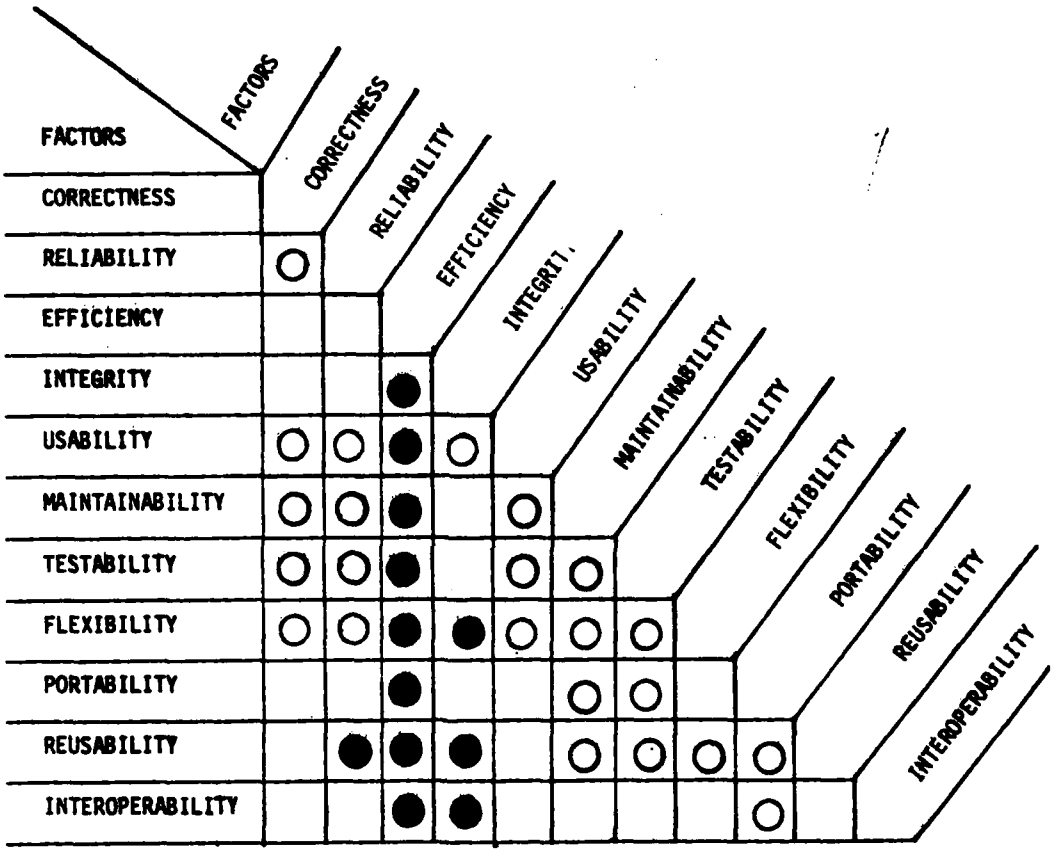
Usability vs Efficiency - the additional code and processing required to ease an operator's task or provide more usable output usually lengthen run time and require additional storage.

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.

Table 4.2-3 Relationships Between Software Quality Factors



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

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 protocol and 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 software which compounds the security problems as well.

SECTION 5

EXAMINATION OF SOFTWARE PRODUCTS THROUGHOUT THE LIFE CYCLE PHASES

5.1 SOFTWARE PRODUCTS AS SOURCES FOR METRICS

One of our main considerations in establishing metrics for the criteria defined in the previous section is the availability of data or software products which provide the sources for the collection of the metrics. Software products include the source code (the most obvious and researched source of metrics to date), documentation including requirements specifications, design specifications, manuals, test plans, problem reports and correction reports, and reviews.

The most accurate source of measures of the qualities of a software product is naturally its operational history. If after two years of operation a software product must be converted to a new hardware system and the cost to accomplish this is 100 % of the initial development cost, it can be stated that from a portability aspect, this product was of poor quality. Testing provides very quantitative measures of the qualities of a software product. The completeness of the testing has always been a concern though. Most testing is oriented toward insuring the software product runs as efficiently as necessary, performs functionally well (correctness), and does not fail (reliability). Extensive testing usually also reveals the usability and testability of the product although not often with the people who will be using the system or testing changes to it. Under special circumstances tests may be oriented toward evaluating the integrity of a product. In most cases though, with the pressures of tight budgets and schedules, testing is never as thorough as desired. Even when it reveals problems, the costs to correct those problems (often involving redesign as well as recoding and retesting) are very high.

The importance of finding errors earlier in the life cycle is well known, since the cost to fix an error increases rapidly as the life cycle progresses. This is shown in Figure 5.1-1. In the same manner, detection of unacceptable quality early is critical to achieving a high quality end product.

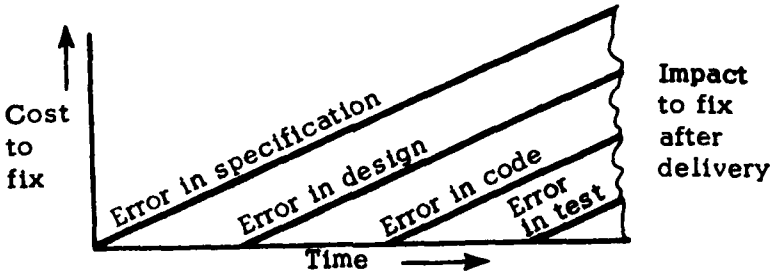


Figure 5.1-1. Impact of Error

It is the intent of this study to concentrate on the early phases of the development life cycle for metrics which will provide an indication of the progression toward the desired product quality. The earlier in the life cycle the more "indicative only" these metrics will be. Obviously if a design specification is perfectly written but poorly implemented the resulting software product will not exhibit a high level of quality. Metrics collected during the implementation phase will help prevent or detect a poor implementation. This concept of successive application of metrics is explained further in Section 6. These concepts are illustrated in Figure 5.1-2.

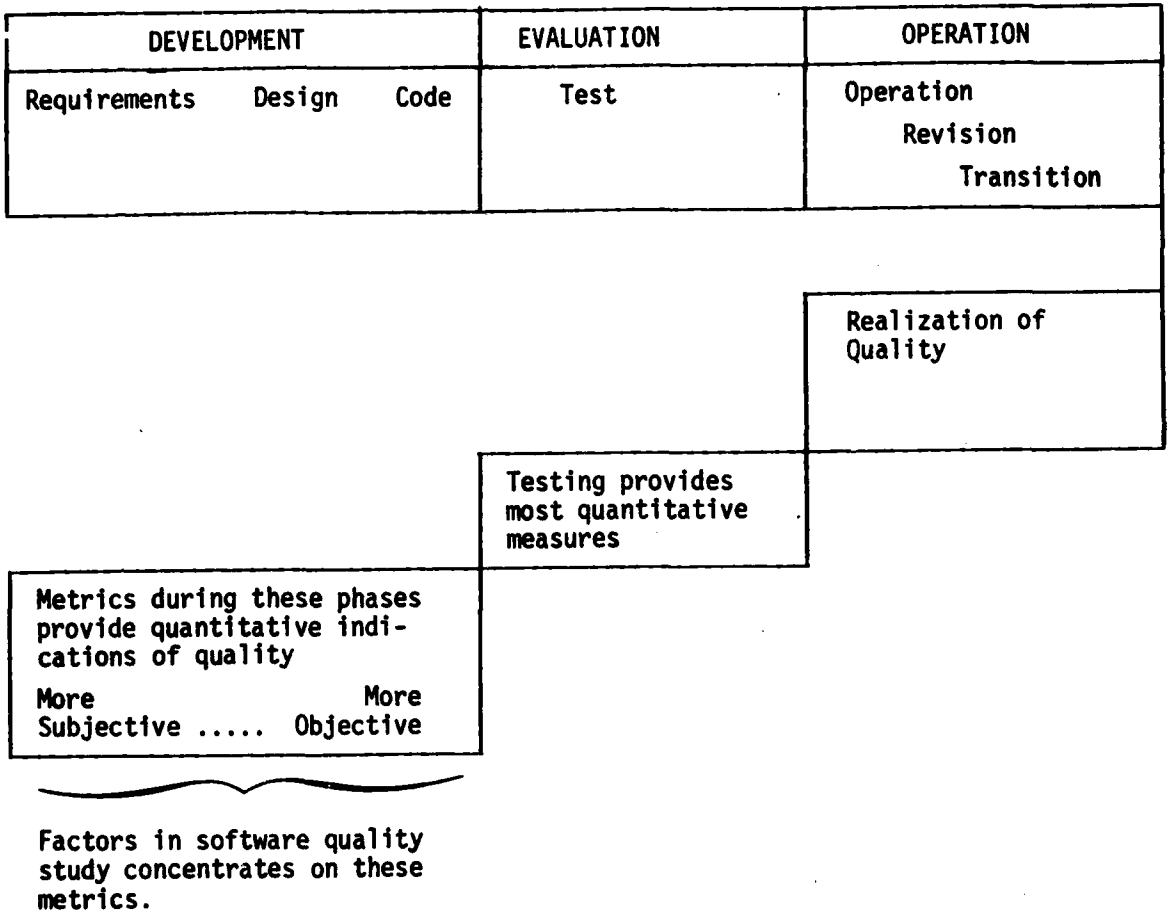


Figure 5.1-2 Concept of Metrics

5.2 RANGE OF SOFTWARE PRODUCTS

While the source code is the primary source for metrics during the development phase, documentation and reviews are generally available earlier and if available provide sources for metrics. The results of simulations at the requirements analysis and design stages are extremely valuable indicators of the quality of the final product. However, because simulation is more of a tool to aid in the development and testing (functional validation) of the requirements and design and is not universally used, it is considered to be out of the scope of this study and not a source for formal metrics.

The range of documents varies widely between SPO's and applications. Several standard documents are required by DOD/AF regulations. The following references were used to compile the range of documents identified in Table 5.2-1.

BOEHB73 — Boehm, B., et al, "Characteristics of Software Quality", Document #25201-6001-RU-00, NBS Contract #3-36012, 28 December 73.

COMP69 — "Computer Program Development and Configuration Management", AFSCF Exhibit 375-2, March 69.

COMP66a — "Computer Program Development and Configuration Management for the Manned Orbit Laboratory Program", SAFSL Exhibit 20012, September 66.

COMP66b — "Computer Program Subsystem Development Milestones", AFSCF SSD Exhibit 61-47B, April 66.

CONF64 — "Configuration Management during Definition and Acquisition Phases", AFSCM 375-1, June 64.

CONF66 — "Configuration Management of Computer Programs", ESD Exhibit EST-1, Section H, April 66.

DOCU74 — "Documentation Standards", Structured Programming Series Vol. VII and Addendum, RADC-TR-74-300, September 74 and April 75.

DODM72 — "DOD Directive for Automation, Policy, Technology, and Standard", DOD Manual 4120.17-M, December 72.

HAGAS75 — Hagan, S., "An Air Force Guide for Monitoring and Reporting Software Development Status," NTIS AD-A016488, September 75.

Table 5.2-1 Reference Documents

	LONG LIFE/HIGH COST SOFTWARE SYSTEMS	SHORT LIFE/LOW COST SOFTWARE SYSTEMS
Specifications	System Requirements Specification Standards & Conventions Documentation Plan Data Base Management Plan Preliminary Design Specification Interface Control Document Preliminary Design Review Material Detailed Design Spec (Build to) Critical Design Review Material Detailed Design Spec (Built to) Validation and Acceptance Test Plan Operator Interface Document	Preliminary Design Spec Detailed Design Spec (Build to) Detailed Design Spec (Built to) User's Guide
Configuration Control Forms	Design Problem Report Software Problem Report Documentation Update Transmittal Compool Change Request Data Base Change Request Modification Transmittal Memo	Revised Version Document

MILI70	"Military Standard Configuration Management Practices for Systems, Equipment, Munitions and Computer Programs", MIL-STD-483, December 70.
MILI68	"Military Standard Specification Practices", MIL-STD-490, October 68.
PILIM68	Piligian, M.S., et al, "Configuration Management of Computer Program Contract End Items", ESD-TR-68-107, January 68.
SCHOW76	Schoeffel, W., "An Air Force Guide to Software Documentation Requirements," NTIS AD-A027 051, June 76.
TACT74	"Tactical Digital Systems Documentation Standards," Department of the Navy, SECNAVINST 3560.1, August 74.

The table illustrates the considerable difference in the quantity of documents among different software system developments. The documents are specified by the AF regulations or SPO-local regulations listed above.

Each of the document types for a long life/high cost software system are characterized briefly in Appendix B. Figure B-1 identifies the documents on a software development timeline that were developed for the two software systems which were utilized to validate the metrics. This figure identifies the most comprehensive set of documents required during software developments. Table B-1 identifies which documents are required by or described in each of the references.

In order to make the metrics widely applicable, a representative set of documents (sources for metrics) for any software development had to be chosen. It was decided that there are basic documentation requirements regardless of the size or cost of the project. Certainly low cost/short life projects incorporate several of the above document characteristics into one document. The documents may not be in as much detail. They are, nevertheless, desirable because of their contribution to a high quality product. Thus while the quantity of documents may vary with the size of the system development effort, the documentation should provide the information contained in the documentation summarized in Appendix B. If it does not,

less information is known about the system, how it is progressing, and later how others will maintain, change, move it, etc. The metrics described in the next section are not based on the availability of specific documents but rather on the availability of the information contained in the documents.

SECTION 6

DEFINITIONS OF METRICS

6.1 DEVELOPMENT OF METRICS

The criteria were defined as the attributes of the software which provide or determine a characteristic of the final software product. Metrics are defined to provide a measure of these attributes. Where more than one attribute or source for metrics are found for any one criterion, subcriteria are established. The subcriteria further clarify the metric and maintain a one-to-one relationship with the metrics. This further enhances the hierarchial nature of the framework for factors in software quality.

Essentially, there are two types of metrics and both are utilized in this study. The first type, like a ruler, is a relative quantity measure. The second type is a binary measure which determines the existence (1) or absence (0) of something. The units of a metric are important to avoid ambiguity and obtain a meaningful metric. The following rule was used in choosing the units of a metric.

> THE UNITS OF THE METRIC WILL BE CHOSEN AS THE RATIO OF ACTUAL OCCURRENCES TO THE POSSIBLE NUMBER OF OCCURRENCES.

Once stated, this rule seems obvious, yet many studies have failed because they did not comply with this rule. To not comply results in poor correlation between the criterion and the factor. To illustrate the types of metrics and the application of the above rule, the following two elementary examples are provided.

(1) The subcriterion, number of unconditional branches, could relate to the criterion, simplicity. Figure 6.1-1 identifies potential metrics for this subcriterion. The metric, the number of GOTO statements/executable statements is the most unambiguous because GOTO statements are a proper subset of the set of executable statements. Blank cards, comment cards, and declarative statements are

```
1         SUBROUTINE TABSCH(NSYM,I,NFLAG)
2   C
3         COMMON/POLTAB/NPT
4         DIMENSION NPT(200)
5   C
6   C        SEARCH THRU DATA BLOCK NPT FOR THE LENGTH UNIT
7   C        OF THE SYMBOL
8   C
9
10        NFLAG = 0
11        MAXSYM = 200
12  C
13        DO 200 J = I,MAXSYM
14  C
15        IF (NPT(J) .EQ. 0) GOTO 300
16        IF (NSYM .EQ. NPT(J)) GOTO 100
17  C
18        J= J+2*NPT(J+1)+4
19        GOTO 200
20  C
21  100   I = J
22        NFLAG = 1
23        J = MAXSYM
24  C
25  200   CONTINUE
26
27  300   RETURN
28        END
```

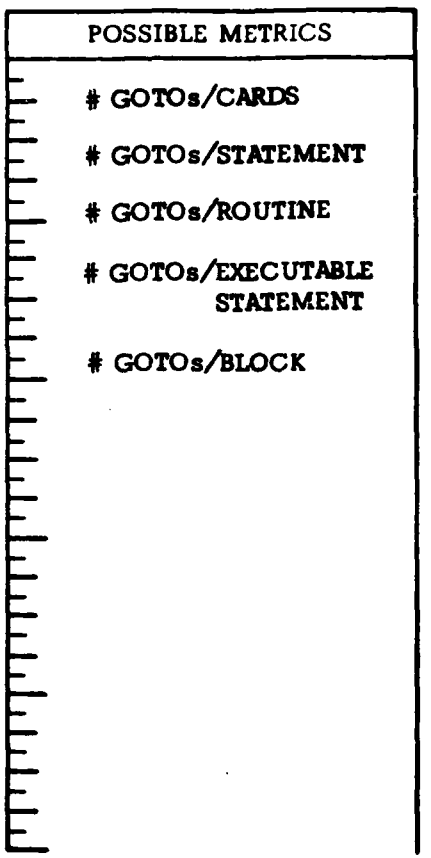


Figure 6.1-1 Choosing a Metric

not potential GOTO statements so that the metrics, GOTO statements/cards and GOTO statements/statements, invite ambiguities to the correlation of GOTO statements with the associated quality factor. GOTO statements/routine and GOTO statements/block do not reveal the size of the set of possible occurrences and, therefore, are ambiguous themselves. This is an example of the first type of metric which provides a relative quantity as a measure. It also provides a normalization of the metric so that any measurement is between 0 and 1.

(2) The metric, use of structured code, is an example of a binary metric and will be given a 1 if present; i.e., the target program utilizes structured coding techniques or a structured preprocessor and, if absent, a 0. The stated rule still holds in these cases as the presence of structured code/possible occurrence of structure code can be viewed as a 1/1 = 1 case. The binary case is to be used where ambiguity or subjectivity may enter the measurement. In the example discussed, measuring the degree to which the code is structured would be subjective and not provide any enhancement to the relationship of structured code to the associated quality factor. A binary metric can also be used to identify an attribute (criteria) which if missing even once in a module or system is very detrimental to the resulting quality of the software product.

Thus both type metrics are consistent with the rule for choosing units. In addition, the metrics have been chosen to be as objective in nature as possible. There are a few exceptions. In these cases, we have attempted to make the subjective metric as quantitative and simple to apply as possible.

The metrics were also chosen to be language independent. The above two examples are consistent with this rule. If a structured language which did not have a GOTO-like construct was utilized both metrics would be 1 indicating the advantage of using that language.

Our process of developing metrics involved the following steps:

(1) Incorporation of work sponsored by RADC in the areas of reliability ([THAYT76], [SHOOM75], [SULLJ73]), portability ([MEALG68], [MARSS70]) and maintainability as well as other significant efforts in the areas of metrics ([BOEHB73], [ELSHJ76], [KOSAS74], [RUBER68], [GILBT76]) and software design ([YOURE75], [MYERG75], [MYERG76], [VANTD74], [KERNB74]). Other references are in the Reference Section.

(2) Incorporation of metrics (oriented primarily to source code and configuration management) which our management have been using for several years.

(3) Application of research in complexity measures [RICHP76] and software physics ([LOVET76a], [FITZA76]) that we have conducted in the past several years and has been conducted by others ([DUNSH77], [ELSHJ76], [HALSM72]).

(4) Evaluation of all of the software products available during a software development (Section 5) with a more global view of software quality provided by the framework established in the first two phases of this report.

(5) Utilization of measures or data available from state-of-the-art software support tools applicable during the requirements analysis and design phases of development ([BROON76], [DAVIC76], [NBS74], [PANZD76], [NODA75], [REIFD76], [CHANP76], [RICHP74], [TIECD76]).

(6) Application of the described concepts of a metric.

The metrics established are explained in paragraph 6.2. Not all of the metrics identified were supported by the validation. However, since the validation was performed with a limited sample and further evaluation and experience is required, the entire set of metrics established are presented. (For further discussion, see Volume II.)

6.2 DESCRIPTION OF METRICS

6.2.1 IDENTIFICATION OF METRICS

G. Myers [MYERG76] states that the "single major cause of software errors is mistakes in translating information." TRW [THAYT76], through an extensive analysis of error data, states that most "errors were design and requirements errors, as opposed to coding errors and errors made during the correction of other errors." An internal survey we conducted substantiated these points [LOVET76]. The documents and reviews discussed in section 5 represent the translation steps from users needs and desired qualities to system implementation; requirements analysis, design, implementation. Our metrics are oriented toward these steps. Since the metrics are indicators of the progression toward good quality, we found that in some cases a metric would be applied at all three phases of development. Each measurement constitutes a unique metric. Thus, during the requirements phase, collection of a set of metrics will provide a measure of the progression toward the desired levels of quality at that point in time. The metrics established are identified in this manner (according to phases) and in relationship to the criteria/subcriteria in Table 6.2-1.

The metrics can be applied at two levels in most cases, at the module level or the system level. The SPO will utilize the metrics at the system level to obtain an overall measure of how the system is progressing with respect to a particular quality factor. At the system level the SPO will be interested in which metrics or metric elements scores are unusually low. It may be there is a general failure to comply with a standard by all designers. Corrective action could be taken to alleviate this type of problem. The SPO may also be interested in a module by module metric value. A particular metric's low score at a system level may not be caused by a general failure but rather by particular modules having unusually low scores. In this case the SPO would want to be aware of which modules are problems so that particular corrective actions or more emphasis could be placed on the development of those modules. The metrics which cannot be applied at both levels are noted in the table.

To illustrate how this table should be read, a few examples will be discussed. The first metric identified in the table corresponds to the criterion, traceability. The metric is the number of itemized requirements traced divided by the total number of itemized requirements. This metric has significance and can be measured at two points in time, during the design and implementation phases. During the design phase, the identification of which itemized requirements are being satisfied in the design of a module should be documented. A traceability matrix is one way of providing this trace. During the implementation phase, identification of which portions of code implement each itemized requirement should be made. Some form of automated notation, prologue comments or imbedded comments, is commonly used to provide the trace at this level. The bold line boxes under design and implementation value columns represent these two measurements. The value columns are used because both measurements are relative quantity or calculated; the ratio of the itemized requirements traced to total number of requirements. A metric which is a binary measure would be placed in a bold line box in the yes/no column. An example of a binary measure is SI.2, the use of a structured language or a structured preprocessor (paragraph 6.2.2.6). A 1 or Ø would be placed in the yes/no column for this metric. Some metrics are checklist type metrics in that there are several elements which must be measured and then summarized/normalized. The summarization/normalization is done in the summary line which is labeled, Metric Value. To be consistent, the final value of each metric is placed in the summary line. The next example will illustrate a combination of these different situations.

The metric associated with the design structure relative to the criterion simplicity (see page 6-12) illustrates the various combinations a metric can assume. Several elements make up the metric. These elements are measured in the design and implementation phases. It is possible to have a metric which is only measured during one phase. Some of the elements are applied by a binary measure (yes/no column) and some are applied by a relative quantity measure, in which case the measure is indicated in parentheses. The overall metric value for the system is indicated in the summary line. Explanations of each of the metrics and metric elements can be found in the following paragraphs. Further information and examples are available in Section 8 and Appendix D, where collecting metric data is described.

Table 6.2-1 Software Quality Metrics

FACTOR(S): CORRECTNESS

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

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): CORRECTNESS, RELIABILITY, MAINTAINABILITY

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

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): RELIABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
ACCURACY	AY. 1 ACCURACY CHECKLIST: (1) Error analysis performed and budgeted to module. (2) A definitive statement of requirement for accuracy of inputs, outputs, processing, and constants. (3) Sufficiency of math library. (4) Sufficiency of numerical methods. (5) Execution outputs within tolerances.	☐ ☐		☐ ☐		☐ ☐	
	SYSTEM METRIC VALUE: $\frac{\text{Score total from applicable elements}}{\text{\# applicable elements}}$		☐		☐		☐
ERROR TOLERANCE/ CONTROL	ET. 1 ERROR TOLERANCE CONTROL CHECKLIST: (1) Any concurrent processing centrally controlled. (2) Errors should be fixable and processing continued. (3) When an error condition is detected, it should be passed up to calling routine.			☐ ☐ ☐		☐ ☐ ☐	
	SYSTEM METRIC VALUE: $\frac{\text{Total score from applicable elements}}{\text{\# applicable elements}}$				☐		☐

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): RELIABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
INPUT DATA	ET. 2 RECOVERY FROM IMPROPER INPUT DATA CHECKLIST: (1) A definitive statement of requirement for error tolerance of input data. (2) Range of values (reasonableness) for items specified and checked . (3) Conflicting requests and illegal combinations identified and checked. (4) All input is checked before processing begins (5) Determination that all data is available prior to processing.	☐		☐ ☐ ☐ ☐		☐ ☐ ☐ ☐	
	SYSTEM METRIC VALUE: $\frac{\text{Total score from applicable elements}}{\text{\# applicable elements}}$		☐		☐		☐
RECOVERABLE COMPUTATIONAL FAILURES	ET. 3 RECOVERY FROM COMPUTATIONAL FAILURES CHECKLIST: (1) A definitive statement of requirement for recovery from computational failures. (2) Loop and multiple transfer index parameters range tested before use. $\left(1-\frac{\text{\# modules with violations}}{\text{total \# modules}}\right)$ (3) Subscript checking. $\left(1-\frac{\text{\# modules with violations}}{\text{total \# modules}}\right)$ (4) Critical output parameters reasonableness checked during processing. $\left(1-\frac{\text{\#modules with violations}}{\text{total \# modules}}\right)$	☐		☐ ☐ ☐		☐ ☐ ☐	
	SYSTEM METRIC VALUE: $\frac{\text{Total score from applicable elements}}{\text{\# applicable elements}}$		☐		☐		☐

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): RELIABILITY

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

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): RELIABILITY, MAINTAINABILITY, TESTABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR 0	VALUE	YES/NO 1 OR 0	VALUE	YES/NO 1 OR 0	VALUE
SIMPLICITY/ DESIGN STRUCTURE	SI. 1 DESIGN STRUCTURE MEASURE: (1) Design organized in top down fashion. (2) No duplicate functions. (3) Independence of module. $\left(1 - \frac{\#\text{ modules violate rule}}{\text{total \# modules}}\right)$ (4) Module processing not dependent on prior processing. $\left(1 - \frac{\#\text{ modules violate rule}}{\text{total \# modules}}\right)$ (5) Each module description includes input, output processing, limitations. $\left(1 - \frac{\#\text{ modules violate rule}}{\text{total \# modules}}\right)$ (6) Each module has single entrance, single exit. $\left(1 - \frac{\#\text{ modules violate rule}}{\text{total \# modules}}\right)$ (7) No global data.			☐ ☐ ☐ ☐ ☐ ☐ ☐		☐ ☐ ☐ ☐ ☐ ☐ ☐	
	SYSTEM METRIC VALUE: $\frac{\text{Total score from applicable elements}}{\text{\# applicable elements}}$			☐		☐	
STRUCTURED PROGRAMMING	SI. 2 USE OF STRUCTURED LANGUAGE OR PREPROCESSOR Structured language or structured language preprocessor used to implement module. If used = 1, if not used = 0.					☐	
	SYSTEM METRIC VALUE: $\frac{\text{Sum of module scores}}{\text{\# modules}}$						☐

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): CONTINUED

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
DATA AND CONTROL FLOW COMPLEXITY	SI. 3 COMPLEXITY MEASURE (by module, see para. 6.2.2.6)				☐		☐
	SYSTEM METRIC VALUE: $\frac{\text{Sum of complexity measures for each module}}{\text{\# modules}}$				☐		☐
CODE SIMPLICITY	SI. 4 MEASURE OF CODING SIMPLICITY (by module)						
	(1) Module flow top to bottom.					☐	
	(2) Negative Boolean or complicated compound Boolean expressions used. $\left(1 - \frac{\text{\# of above}}{\text{\# executable statements}}\right)$						☐
	(3) Jumps in and out of loops. $\frac{(\text{\# single entry/single exit loops})}{\text{total \# loops}}$						☐
	(4) Loop index modified. $\left(1 - \frac{\text{\# loop indices modified}}{\text{total \# loops}}\right)$						☐
	(5) Module is not self-modifying.					☐	
	(6) All arguments passed to a module are parametric.					☐	
	(7) Number of statement labels. $\left(1 - \frac{\text{\# labels}}{\text{\# executable statements}}\right)$						☐
	(8) Unique names for variables.					☐	
	(9) Single use of variables.					☐	
	(10) No mixed mode expressions.					☐	
	(11) Nesting level. $\left(\frac{1}{\text{max nesting level}}\right)$						☐
	(12) Number of branches. $\left(1 - \frac{\text{\#branches}}{\text{\#executable statements}}\right)$						☐

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): CONTINUED

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR 0	VALUE	YES/NO 1 OR 0	VALUE	YES/NO 1 OR 0	VALUE
	SI. 4 (CONTINUED) (13) Number of GOTOs. $\left(1 - \frac{\text{\# GOTO statements}}{\text{\# executable statements}}\right)$						☐
	(14) No extraneous code exists.					☐	
	(15) Variable mix in a module. $\left(\frac{\text{\# internal variables}}{\text{total \# variables}}\right)$						☐
	(16) Variable density. $1 - \left(\frac{\text{\# variables}}{\text{\# exec statements}}\right)$						☐
	MODULE METRIC VALUE: $\frac{\text{Total score from applicable elements}}{\text{\# applicable elements}}$						☐
	SYSTEM METRIC VALUE: $\frac{\text{Sum of code simplicity measures for each module}}{\text{\# modules}}$						☐

Table 6.2-1 Software Quality Metrics (Continued)

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

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
MODULARITY/ DEGREE OF INDEPENDENCE	MO. 1 STABILITY MEASURE $\left(\frac{\text{Expected \# modules changed}}{\text{total \# modules}}\right)$				☐		
	SYSTEM METRIC VALUE: Same as entry above				☐		
MODULAR IMPLEMENTATION	MO. 2 MODULAR IMPLEMENTATION MEASURE						
	(1) Hierarchical structure. $\left(1-\frac{\text{\# violations of hierarchy}}{\text{total \# modules}}\right)$				☐		☐
	(2) All modules do not exceed standard module size (100). $\left(1-\frac{\text{\# modules} > 100}{\text{total \# modules}}\right)$						☐
	(3) All modules represent one function. $\left(1-\frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$				☐		☐
	(4) Controlling parameters defined by calling module. $\left(1-\frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$				☐		☐
	(5) Input data controlled by calling module. $\left(1-\frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$				☐		☐
	(6) Output data provided to calling module. $\left(1-\frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$				☐		☐
	(7) Control returned to calling module. $\left(1-\frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$				☐		☐
	(8) Modules do not share temporary storage.			☐		☐	
	SYSTEM METRIC VALUE $= \left(\frac{\text{Total score from applicable elements}}{\text{\# applicable elements}}\right)$				☐		☐

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): FLEXIBILITY, REUSABILITY

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

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): FLEXIBILITY

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

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): TESTABILITY

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

Table 6.2-1 Software Quality Metrics (Continued)

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

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

Table 6.2-1 Software Quality Metrics (Continued)

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

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

Table 6.2-1 Software Quality Metrics (Continued)

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

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

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): EFFICIENCY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
EXECUTION EFFICIENCY/ REQUIREMENTS	EE. 1 PERFORMANCE REQUIREMENTS ALLOCATED TO DESIGN			☐			
	SYSTEM METRIC VALUE = Same as line above				☐		
ITERATIVE PROCESSING	EE. 2 ITERATIVE PROCESSING EFFICIENCY MEASURE: (by module)						
	(1) Non-loop dependent computations kept out of loop. $\left(1 - \frac{\#\text{ nonloop dependent statements in loop}}{\text{total }\#\text{ loop statements}}\right)$				☐		☐
	(2) Performance optimizing compiler/assembly language used.					☐	
	(3) Compound expressions defined once. $\left(1 - \frac{\#\text{ compound expression defined more than once}}{\#\text{ compound expressions}}\right)$						☐
	(4) Number of overlays. $\left(\frac{1}{\#\text{ of overlays}}\right)$				☐		☐
	(5) Free of bit/byte packing/unpacking in loops.			☐		☐	
	(6) Free of nonfunctional executable code. $\left(1 - \frac{\#\text{ nonfunctional executable code}}{\text{total executable statements}}\right)$						☐
	(7) Decision statements efficiently coded. $\left(1 - \frac{\#\text{ inefficient decision statements}}{\text{Total }\#\text{ decision statements}}\right)$						☐

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): EFFICIENCY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
	(8) Module linkages. $\left(1 - \frac{\text{module linkage time}}{\text{execution time}}\right)$ (9) OS linkages. $\left(1 - \frac{\text{OS linkage time}}{\text{execution time}}\right)$						☐ ☐
	MODULE METRIC VALUE = $\frac{\text{total score from applicable elements}}{\text{total \# applicable elements}}$				☐		☐
	SYSTEM METRIC VALUE = $\frac{\text{sum of iterative processing measures for each module}}{\text{total \# modules}}$				☐		☐
DATA USACE	EE. 3 DATA USAGE EFFICIENCY MEASURE: (by module)						
	(1) Data grouped for efficient processing.			☐		☐	☐
	(2) Variables initialized when declared. $\left(\frac{\text{\# initialized when declared}}{\text{total \# variables}}\right)$						☐
	(3) No mix-mode expressions. $\left(1 - \frac{\text{\# mix mode expressions}}{\text{\# executable statements}}\right)$						☐
	(4) Common choice of units/type. $\left(1/\text{\# occurrences of uncommon unit operations}\right)$				☐		☐
	(5) Data indexed or referenced for efficient processing.			☐		☐	
	MODULE METRIC VALUE = $\frac{\text{total score from applicable elements}}{\text{\# applicable elements}}$				☐		☐
	SYSTEM METRIC VALUE = $\frac{\text{sum of data usage measures for each element}}{\text{total \# modules}}$				☐		☐

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): EFFICIENCY

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

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): INTEGRITY

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

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): USABILITY

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

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): USABILITY

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

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): USABILITY

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

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): PORTABILITY, REUSABILITY

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

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): PORTABILITY, REUSABILITY

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

Table 6.2-1 *Software Quality Metrics (Continued)*

FACTOR(S): INTEROPERABILITY

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

Table 6.2-1 Software Quality Metrics (Continued)

FACTOR(S): MAINTAINABILITY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
CONCISENESS	CO. 1 HALSTEAD'S MEASURE (by module) $\left(1 - \frac{\text{module length calculated-module length observed}}{\text{module length observed}}\right)$						☐
	SYSTEM METRIC VALUE = $\frac{\text{Sum Halstead's measure for each module}}{\text{total \# modules}}$						☐

6.2.2 EXPLANATIONS OF METRICS

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

6.2.2.1 Traceability

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

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

6.2.2.2 Completeness

CP.1 Completeness Checklist (All three phases at system level).
This metric is the sum of the scores of the following applicable elements divided by the number of applicable elements.

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

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

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

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

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

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

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

(8) Design agrees with requirements.
Continual updating of the requirements documentation and the design documentation is required so that the current version of the source code (see element (9)), the current version of the design documentation, and the current version of the requirements documentation agreee.

(9) Code agrees with design.
See element (8).

6.2.2.3 Consistency

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

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

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

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

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

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

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

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

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

(3) Unit Consistency.
Units of variables are to be chosen to be consistent with all uses of the variables. The measure is based on the number of modules in which consistent units are not utilized. This can be measured at both design and implementation.

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

(5) Data Type Consistency.
A data element defined as a particular data type is to be used as that data type in all occurrences. A common violation of this rule is found in arrays where several data types are defined. The measure is based on the number of modules which utilize data types inconsistently.

6.2.2.4 Accuracy

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

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

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

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

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

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

6.2.2.5 Error Tolerance

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(2) Recovery from Hardware Faults (design and implementation phases only).
The design specification and code to provide the recovery from the hardware faults identified in the requirements must exist in the design and implementation phases respectively.

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

(1) A definitive statement of requirements for recovery from device errors (requirements only).
The handling of device errors such as unexpected end-of-files or end-of-tape conditions or read/write failures are specified during the requirements phase.

(2) Recovery from Device Errors (design and implementation phases only).
The design specification and code to provide the required handling of device errors must exist in the design and implementation phases respectively.

6.2.2.6 Simplicity

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

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

(2) No duplicate functions.
Descriptions of functions to be performed by each module at design and the actual function performed by the coded module is to be evaluated to ensure it is not duplicated by other modules.

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

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

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

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

(7) No global data.
This is a binary measure which identifies the complexity added to a system by the use of global data. If no global data exists, this measure is 1, if global data does exist, it is 0.

SI.2 Use of structured language or structured language preprocessor (implementation phase). The metric is a binary measure of the existence (1) or absense (0) of structured language code.
A structured language or structured language preprocessor provides constructs similar to the IFTHENELSE, DOWHILE, DOUNTIL, and CASE statements associated with structured programming.

SI.3 Data and Control Flow Complexity measure (Design and implementation phases).
This metric can be measured from the design representation (e.g., flowcharts) and the code automatically. Path flow analysis and variable set/use information along each path is utilized. A variable is considered to be 'live' at a node if it can be used again along that path in the program. The complexity measure is based on summing the 'liveness' of all variables along all paths in the program. It is normalized by dividing it by the maximum complexity of the program (all variables live along all paths).
(See [RICHP76] and page D-16 of Volume II.)

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

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

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

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

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

(5) Module is not self-modifying.
If a module has the capability to modify its processing logic it becomes very difficult to recognize what state it is in when an error occurs. In addition, static analysis of the logic is more difficult. This measure emphasizes the added complexity of self-modifying modules.

(6) All arguments passed to a module are parametric.
This is a binary measure, 1 if all parameters are parametric, 0 if they all are not. This measure is based on the potential problems that can arise if constants or global data are used as arguments.

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

(8) Unique names for variables.
This is a binary measure which is given a 1 if unique names are used, and a 0 if they are not.

(9) Single use of variables.
A variable is to be utilized for only one purpose, i.e., in one manner. This measure is a binary measure; 1 if variables are used in only one way and 0 if they are used for multiple purposes.

(10) No mixed-mode expressions.
If mix-mode expressions are used greater complexity is introduced. This measure is a 1 if no mix-mode expressions are used in a module, and a 0 if mix-mode expressions are used.

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

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

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

(14) No extraneous code exists.
This is a binary measure which is 1 if no extraneous code exists and 0 if it does. Extraneous code is code which is nonfunctional or cannot be executed.

(15) Variable mix in a module.
From a simplicity viewpoint, local variables are far better than global variables. This measure is the ratio of internal (local) variables to total (internal (local) plus external (global)) variables within a module.

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

6.2.2.7 Modularity

MO.1 Stability Measure (Design phase at system level).
This measure is based on G. Meyer's ([MYERG76]) categorization of modules by their module strength and coupling. Module strength is a measure of the cohesiveness or relationship of the elements within a module. Module coupling is a measure of the relationship between modules. The metric combines these two measures to calculate the expected number of modules that would require modification if changes to any one module were made divided by the total number of modules.

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

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

(2) All modules do not exceed a standard module size (100) (Implementation phase only).
The standard module size of 100 procedural statements can vary. 100 was chosen because it has been mentioned in the literature frequently. This measure is based on the number of modules which exceed the standard size established.

(3) All modules represent one function.
The concept of modularity is based on each function being implemented in a unique module. This measure is based on the number of modules which represent more than one function. This can be determined at both design and implementation.

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

(5) Input data controlled by calling module.
See (4) above.

(6) Output data provided to calling module.
See (4) above.

(7) Control returned to calling module.
See (4) above.

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

6.2.2.8 Generality

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

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

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

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

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

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

measure is based on the number of modules which are data value limited. This can be determined at design and implmentation.

(5) All constants should be defined once.
This element, in effect, defines a constant as a parametric value. At one place in the module or data base it can be changed to accommodate a different application of the function of that module, e.g., to calculate a mathematical relationship at a greater degree of precison or to represent the constant of gravitation of a different planet than earth, etc. Thus, if this rule is followed, the effort required to apply the module in a different environment is smaller. The measure is based on the number of modules which violate this concept during design and implementation.

6.2.2.9 Expandability

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

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

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

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

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

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

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

6.2.2.10 Instrumentation

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

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

(2) Input parameters boundary tested.
The other aspect of module testing involves testing the input

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

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

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

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

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

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

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

6.2.2.11 Self Descriptiveness

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

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

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

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

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

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

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

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

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

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

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

(2) Standard format for organization of modules followed.
A specific format ordering such as prologue comments, declarative statements, executable statements is to be uniformly used in modules. This measure is based on the number of modules which do not comply with the standard format established.

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

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

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

(6) No language keywords used as names.
Some languages allow keywords to be used as statement labels or as variables. This practice is confusing to a reader. The measure is based on the number of modules in which a keyword is used in this manner.

6.2.2.12 Execution Efficiency

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

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

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

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

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

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

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

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

(6) Free of nonfunctional executable code (implementation only).
Segments of executable code which do not perform a relevant function are obvious inefficiencies. They arise most often during redesign or editing when updates are made without complete removal of obsolete code. This element can be measured at implementation only and is based on the number of nonfunctional, yet executable lines of code.

(7) Decision Statements efficiently coded (implementation only).
This measure is based on the number of inefficiently coded decision statements divided by the total number of decision statements. An example of an inefficiently coded decision statement is not having the most frequently exercised alternative of an IF statement be the THEN clause.

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

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

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

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

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

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

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

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

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

6.2.2.13 Storage Efficiency

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

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

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

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

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

(5) Data Segmentation.
The amount of data referenced by a module in the form of arrays, input buffers, or global data, often is small compared to the size of the storage areas required. This represents an inefficient use of storage. The measure is based on the amount of unused data divided by the total amount of data available to a module.

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

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

(8) Free of nonfunctional code (implementation only).
Nonfunctional code, whether executable (see EE.2 (6)) or not, consumes storage space so it is undesirable. This measure is based on the number of lines of code which are nonfunctional.

(9) No duplicate code.
Duplicate code should be avoided for the same reason as (8) above. This measure which is to be applied during design and implementation is based on the amount of duplicate code.

(10) Storage optimizing compiler/assembly language used (implementation only).
This binary measure is similar to EE.2 (2) except from the viewpoint of storage optimization.

(11) Free of redundant data elements (implementation only).
This measure pertains to the data base and is based on the number of redundant data elements.

6.2.2.14 Access Control

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

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

(2) Data Base Access controls provided.
This binary measure identifies whether requirements for data base

controls have been specified, designed and the capabilities implementated. Examples of data base access controls are authorization tables and privacy locks.

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

6.2.2.15 Access Audit

AA.1 Access Audit Checklist (all three phases at system level).
The metric is the averaged score of the following two elements.

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

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

6.2.2.16 Operability

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

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

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

(3) Provisions for operator to interrupt, obtain status, save, modify, and continue processing.
The capabilities provided to the operator must be considered during the requirements phase and then designed and implemented. Examples of operator capabilities include halt/resume and check pointing. This is a binary measure to trace the evolution of these capabilities.

(4) Number of operator actions reasonable (implementation only, requires execution).
The number of operator errors can be related directly to the number of actions required during a time period. This measure is based on the amount of time spent requiring manual operator actions divided by the total time required for the job.

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

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

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

6.2.2.17 Training

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

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

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

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

6.2.2.18 Communicativeness

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.2.2.19 Software System Independence

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

(1) Dependence on Software System Utility programs.
The more utility programs that are used within a system the more dependent the system is on that software system environment. A

SORT utility in one operating system is unlikely to be exactly similar to a SORT utility in another. This measure is based on the number of programs used that are utility programs divided by the total number of programs in the system. It is to be applied during design and implementation.

(2) Dependence on Software System Library Routines.
For similar reasons as (1) above an integration function provided by one operating system may not be exactly the same as an integration function provided by another system. Thus the more library routines used the more dependent the system is on its current software system environment. This measure, applied at design and implementation, is based on the number of library routines used divided by the total number of modules in the system.

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

(4) Free from Operating System References.
This measure is based on the number of modules which contain calls to the operating system. While (1) and (2) above identify the number of support-type programs and routines which might have to be recoded if a change in software system environments took place, this measure identifies the percent of application oriented modules which would probably have to be changed. The metric is to be applied during design and implementation.

6.2.2.20 Machine Independence

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

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

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

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

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

6.2.2.21 Communications Commonality

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

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

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

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

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

6.2.2.22 Data Commonality

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

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

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

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

6.2.2.23 Conciseness

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

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

The calculated length of a module is

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

The metric is normalized as follows:

$$1 - \frac{|N_C - N_O|}{N_O} \text{ or,}$$

$$0 \text{ if } \frac{|N_C - N_O|}{N_O} \text{ greater than } 1$$

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

6.3 SUMMARIZATION OF METRICS

Table 6.3-1 provides summary figures for the metrics that have been established.

41 metrics have been established for the 11 factors and 23 criteria. These 41 metrics are comprised of 175 elements or specific characteristics of a software product that can be measured at various times during development to give an indication of the progression toward a desired level of quality. The metrics are applied during the three phases of development as shown. Thus 25 characteristics of the software product are measured during the requirements analysis phase, 108 during the design phase, and 157 during implementation.

Table 6.3-1 Summarization of Metrics

NO. OF FACTORS	NO. OF CRITERIA	NO. OF SUBCRITERIA	NO. OF METRICS	NO. OF ELEMENTS
11	23	28	41	175

NO. OF METRICS APPLIED DURING:	ALL 3 PHASES	2 PHASES	1 PHASE
	13	20	8

PHASE COVERAGE	REQUIREMENTS	DESIGN	IMPLEMENTATION
No. of Metrics applied during:	13	35	39
No. of Elements applied during:	25	108	157

REFERENCES

ABERD72 Abernathy, D.H., et al, "Survey of Design Goals for Operating Systems", Georgia Tech, GITIS-72-04, 1972.

ACQU71 "Acquisition and Use of Software Products for Automatic Data Processing Systems in the Federal Government", Comptroller General of the U.S., Report to the Congress, June 1971.

AIRF76 "Air Force Systems Command", Aviation Week & Space Technology, 19 July 1976.

ALGEC77 Algea, C., "ATP - Analysis of JOVIAL (J4) Routines", Internal GE Working Paper, March 1977.

AMORW73 Amory, W., Clapp, J.A., "An Error Classification Methodology", MITRE Tech Report, June 1973.

BELLD74 Bell, D.E., Sullivan, J.E., "Further Investigations into the Complexity of Software", MITRE Tech Report MTR-2874, June 1974.

BELLT76 Bell, T., et al, "An Extendable Approach to Computer-Aided Software Requirements Engineering", 1976 Software Engineering Conference.

BENSJ76 Benson, J., "Some Observations Concerning the Structure of FORTRAN Programs", International Symposium on Fault Tolerant Computing, Paris, June 1975.

BOEHB73a Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., MacLeod, G.S., Merritt, N.J., "Characteristics of Software Quality", Doc. #25201-6001-RU-00, NBS Contract #3-36012, 28 December 1973.

BOEHB76 Boehm, B., Brown, J., Lipow, M., "Quantitative Evaluation of Software Quality", 1976 Software Engineering Conference.

BOEHB73b Boehm, B.W., "Software and its Impact: A Quantitative Approach", Datamation, April 1973.

BOLEN76 Bolen, N., "An Air Force Guide to Contracting for Software Acquisition", NTIS AD-A020 444, January 1976.

BOULD61 Boulanger, D.G., "Program Evaluation and Review Technique", Advanced Management, July-August 1961.

BRADG75 Bradley, G.H., et al, "Structure and Error Detection in Computer Software", Naval Postgraduate School, NTIS AD-A014 334, February 1975.

BROON76 Brooks, N., et al, "Jovial Automated Verification System (JAVS)", RADC-TR-20, February 1976.

BROWJ73 Brown, J.R. and Buchanan, H.N., "The Quantitative Measurement of Software Safety and Reliability", TRW Report SS-73-06, August 1973.

BROWP72 Brown, P., "Levels of Language for Portable Software", Communications of the ACM, December 1972.

CASEJ74 Casey, J.K., "The Changing Role of the In-House Computer Application Software Shop", GE TIS #74AEG195, February 1974.

CHANP76 Chang, P., Richards, P.K., "Software Development and Implementation Aids", GE TIS #76CIS01, January 1976.

CHENL74 Cheng, L., Sullivan, J.E., "Case Studies in Software Design", MITRE Tech Report MTR-2874, June 1974.

CLAPJ74 Clapp, J.A., Sullivan, J.E., "Automated Monitoring of Software Quality", Proceedings from AFIPS Conference, Vol. 43, 1974.

COHEA72 Cohen, A., "Modular Programs: Defining the Module", Datamation, March 1972.

COMP69 "Computer Program Development and Configuration Management", AFSCF Exhibit 375-2, March 1969.

COMP66a "Computer Program Development and Configuration Management for the Manned Orbit Laboratory Program", SAFSL Exhibit 20012, September 1966.

COMP66b "Computer Program Subsystem Development Milestones", AFSCF SSD Exhibit 61-47B, April 1966.

CONF64 "Configuration Management During Definition and Acquisition Phases", AFSCM 375-1, June 1964.

CONF66 "Configuration Management of Computer Programs", ESD Exhibit EST-1, Section H, 1966.

CONNJ75 Connolly, J., "Software Acquisition Management Guidebook: Regulations, Specifications, and Standards", NTIS AD-A016 401, October 1975.

COOLW62 Cooley, T., Multivariate Procedures for the Behavioral Sciences, John Wiley and Sons, Inc., N.Y., 1962.

CORRA74 Corrigan, A.E., "Results of an Experiment in the Application of Software Quality Principles", MITRE Tech Report MTR-2874, June 1974.

CULPL75 Culpepper, L.M., "A System for Reliable Engineering Software", International Conference on Reliable Software, 1975.

DAVIC76 Davis, C., Vick, C., "The Software Development System", 1976 Software Engineering Conference.

DAVIR73 Davis, R.M., "Quality Software can Change the Computer Industry Programs Test Methods", Prentice-Hall, 1973, Chapter 23.

DENNJ70 Dennis, J.B., Goos, G., Poole, J., Gotlieb, C.C., et al, "Advanced Course on Software Engineering", Springer-Verlag, New York 1970.

DIJKE69a Dijkstra, E.W., "Complexity Controlled by Hierarchical Ordering of Function and Variability" Software Engineering, NATO Science Committee Report, January 1969.

DIJKE72 Dijkstra, E.W., "The Humble Programmer", Communications of the ACM, October 1972.

DIJKE69b Dijkstra, E.W., "Structured Programming", Software Engineering Techniques, NATO Science Committee Report, January 1969.

DIJKE72 Dijkstra, E.W., "Notes on Structured Programming", Structured Programming, Dahl, Dijkstra, Hoare, Academic Press, London 1972.

DOCU74 "Documentation Standards", Structured Programming Series Volume VII and Addendum, RADC-TR-74-300, September 1974 and April 1975.

DODM72 "DOD Manual for DOD Automated Data Systems Documentation Standards", DOD Manual 4120.17M, December 1972.

DROSM76 Drossman, M.M., "Development of a Nested Virtual Machine, Data Structure Oriented Software Design Methodology and Procedure for its Evaluation", USAFOSR/RADC Tech Report, 11 August 1976.

DUNSH77 Dunsmore, H., Ganon, J., "Experimental Investigation of Programming Complexity", Proceedings of ACM/NBS Sixteenth Annual Technical Symposium, June 1977.

EDWAN75 Edwards, N.P., "The Effect of Certain Modular Design Principles on Test-ability", International Conference on Reliable Software, 1975.

ELEC75 "The Electronic Air Force", Air Force Magazine, July 1975.

ELSHJ76 Elshoff, J.L., "Measuring Commercial PL/1 Programs Using Halstead's Criteria", SIGPLAN Notices, May 1976.

ELSHJ76b Elshoff, J., "An Analysis of Some Commercial PL/1 Programs", IEE Transactions on Software Engineering, Volume SE-2, No. 2, June 1976.

ENDRA75 Endres, A., "An Analysis of Errors and their Causes in Systems Programs", International Conference on Reliable Software, 1975.

FAGAM76 Fagan, M., "Design and Code Inspections and Process Control in the Development of Programs", IBM TR 00.2763, June 1976.

FIND75 "Findings and Recommendations of the Joint Logistics Commanders", Software Reliability Working Group, November 1975.

FITZA76 Fitzsimmons, A., Love, T., "A Review and Critique of Halstead's Theory of Software Physics", GE TIS #76ISP004, December 1976.

FLEIJ72 Fleiss, J.E., et al, "Programming for Transferability", RADC-TR-72-234, September 1972.

FLEIT66 Fleishman, T., "Current Results from the Analysis of Cost Data for Computer Programming", NTIS AD-637 801, August 1966.

GILBT76 Gilb, T., Software Metrics, Winthrop Computer Systems Series, 1976.

GOODJ74 Goodenough, J., "Effect of Software Structure on Software Reliability, Modifiability, and Reusability: A Case Study", USA Armament Command, March 1974.

GOODJ75 Goodenough, J., "Exception Handling Design Issues", SIGPLAN Notices, July 1975.

GOVE74 "Government/Industry Software Sizing and Costing Workshop-Summary Notes", USAFESD, 1-2 October 1974.

HAGAS75 Hagan, S., "An Air Force Guide for Monitoring and Reporting Software Development Status", NTIS AD-A016 488, September 1975.

HAGUS76 Hague, S.J., Ford, B., "Portability-Prediction and Correction", Software Practices & Experience, Vol. 6, 61-69, 1976.

HALSM77 Halstead, M., Elements of Software Science, Elsevier Computer Science Library, N.Y., 1977.

HALSM73 Halstead, M., "Algorithm Dynamics", Proceedings of Annual Conference of ACM, 1973.

HALSM72 Halstead, M., "Natural Laws Controlling Algorithm Structure", ACM SIGPLAN, February 1972.

HAMIM76 Hamilton, M., Zeldin, S., "Integrated Software Development System/Higher Order Software Conceptual Description", ECOM-76-0329-F, November 1976.

HANEF72 Haney, F.M., "Module Connection Analysis - A Tool for Scheduling Software Debugging Activities", Proceedings of the 1972 Fall Joint Computer Conference, Vol. 41, Part 1, 173-179, 1972.

HODGB76 Hodges, B., Ryan, J., "A System for Automatic Software Evaluation", 1976 Software Engineering Conference.

JONEC77 Jones, C., "Program Quality and Programmer Productivity", IBM TR 02.764, January 1977.

KERNB74 Kernighan, B., Plauger, P., The Elements of Programming Style, McGraw-Hill, 1974.

KESSM70 Kessler, M.M., "An Investigation of Program Structure", IBM Federal Systems Division, Internal Memo, February 1970.

KNUTD68 Knuth, D.E., The Art of Computer Programming Vol. 1, Addison-Wesley, 1968.

KNUTD71 Knuth, D.E., "An Empirical Study of FORTRAN Programs", Software Practice & Experience, Vol. 1, pp 105-133, 1971.

KOSAS74 Kosarajo, S.R., Ledgard, H.F., "Concepts in Quality Software Design", NBS Technical Note 842, August 1974.

KOSYD74 Kosy, D., "Air Force Command and Control Information Processing in the 1980s: Trends in Software Technology", Rand, June 1974.

KUESJ73 Keuster, J., Mize, J., Optimization Techniques with FORTRAN, McGraw-Hill, N.Y., 1973.

LABOV66 LaBolle, V., "Development of Equations for Estimating the Costs of Computer Program Production", NTIS AD-637 760, June 1966.

LAPAL73 LaPadula, L.J., "Software Reliability Modeling and Measurement Techniques", MTR-2648, June 1973.

LARSR75 Larson, R., "Test Plan and Test Case Inspection Specification", IBM TR 21.586, April 1975.

LEWIE63 Lewis, E., Methods of Statistical Analysis, Houghton Mifflin Company, Boston 1963.

LIEBE72 Lieblein, E., "Computer Software: Problems and Possible Solutions", CENTACS USAECOM Memorandum, 7 November 1972.

LIGHW76 Light, W., "Software Reliability/Quality Assurance Practices", Briefing given at AIAA Software Management Conferences, 1976.

LISKB75 Liskov, B., "Data Types and Program Correctness", SIGPLAN Notices, July 1975.

LISKB73 Liskov, B.H., "Guidelines for the Design and Implementation of Reliable Software Systems", MITRE Report 2345, February 1973.

LOVET76a Love, T., Bowman, A., "An Independent Test of the Theory of Software Physics", SIGPLAN Notices, November 1976.

LOVET76b Love, T., Fitzsimmons, A., "A Survey of Software Practioners to Identify Critical Factors in the Software Development Process", GE TIS 76ISP003, December 1976.

MANNJ75 Manna, J., "Logical Analysis of Programs", International Conference on Reliable Software, 1975.

MARSS70 Marshall, S., Millstein, R.E., Sattley, K., "On Program Transferability", Applied Data Research, Inc., RADC-TR-70-217, November 1970.

MCCAT76 McCabe, T., "A Complexity Measure", 1976 Software Engineering Conference.

MCCRD72 McCracken, D.D. and Weinberg, G.M., "How to Write a Readable FORTRAN Program", Datamation, October 1972.

MCKIJ77 McKissick, J., Price, R., "Quality Control of Computer Software", 1977 ASQC Technical Conference Transactions, Philadelphia 1977.

MCNEL75 McNeely, L., "An Approach to the Development of Methods and Measures for Quantitatively Determining the Reliability of Software", Ultra Systems Concept Paper, February 1975.

MEALG68 Mealy, G.H., Farber, D.J., Morehoff, E.E., Sattley, "Program Transferability Study", RADC, November 1968.

MILI70 "Military Standard Configuration Management Practices for Systems, Equipment, Munitions and Computer Programs", MIL-STD-483, December 1970.

MILI68 "Military Standard Specification Practices", MIL-STD-490, October 1968.

MILLE74 Miller, E., et al, "JOVIAL/J3 Automated Verification System (JAVS) System Design Document", GRC, March 1974.

MULOR70 Mulock, R.B., "A Study of Software Reliability at the Stanford Linear Accelerator Center, Stanford University", August 1970.

MYERG73 Myers, G.J., "Characteristics of Composite Design", Datamation, September 1973.

MYERG75 Myers, G.J., Reliable Software through Composite Design, Petrocelli/Charter, 1975.

MYERG76 Myers, G.J., Software Reliability: Principles and Practices, John Wiley & Sons, New York, 1976.

NBS74 "Analyzer - Computation and Flow Analysis", NBS Tech Note 849, 1974.

NELSR74 Nelson, Richard, "A Plan for Quality Software Production", RADC Internal Paper, June 1974.

NELSR75 Nelson, R., Sukert, A., "RADC Software Data Acquisition Program", RADC Paper presented at Fault Tolerant System Workshop, Research Triangle Institute, November 1975.

NODA75 "NODAL - Automated Verification System", Aerospace TOR-0075(5112)-1, 1975.

OGDIJ72 Ogdin, J.L., "Designing Reliable Software", Datamation, July 1972.

OSTEL74 Osterweil, L., et al, "Data Flow Analysis as an Aid in Documentation, Assertion Generation, and Error Detection", NTIS PB-236-654, September 1974.

OSTLB63 Ostle, B., Statistics in Research, Iowa State University Press,

PADED56 Paden, D., Linquist, E., Statistics for Economics and Busin[illegible] Hill, New York, 1956.

PANZD76 Panzl, D., "Test Procedures: A New Approach to Software Verification", 1976 Software Engineering Conference.

PARIR76 Pariseav, R., "Improved Software Productivity for Military Systems through Structured Programming", NTIS AD-A022 284, March 1976.

PARND72a Parnas, D.L., "A Technique for Software Module Specification with Examples", Communications of the ACM, Vol. 15 No. 5, 1972.

PARND71 Parnas, D.L., "Information Distribution Aspects of Design Methodology", Proc IFIP Congress 1971.

PARND75 Parnas, D.L., "The Influence of Software Structure on Reliability", International Conference on Reliable Software, 1975.

PARND72b Parnas, D.L., "On the Criteria to be used in Decomposing Systems into Modules", Comm. of the ACM, Vol. 15, No. 12, December 1972.

PATH76 Pathway Program - Product Quality Assurance for Shipboard Installed Computer Programs, Naval Sea Systems Command, April 1976.

PET72 "PET - Automatic Test Tool", AFIPS Conference Proceedings, Vol. 42, 1972.

PILIM68 Piligian, M.S., et al, "Configuration Management of Computer Program Contract End Items", ESD-TR-68-107, January 1968.

POOLL77 Poole, L., Borchers, M., Some Common Basic Programs, Adam Osborne and Associates, Berkeley, 1977.

PROG75 Program Design Study "Structured Programming Series" (Vol. VIII), RADC TR-74-300, 1975.

RAMAC75 Ramamoorthy, C., Ho, S., "Testing Large Software with Automated Software Evaluation Systems", 1976 Software Engineering Conference.

REIFD75 Reifer, D.J., "Automated Aids for Reliable Software", International Conference on Reliable Software, 1975.

REIFD76 Reifer, D., "Toward Specifying Software Properties", IFIP Working Conference on Modeling of Environmental Systems, Tokyo, Japan, April 1976.

RICHF74 Richards, F.R., "Computer Software Testing, Reliability Models, and Quality Assessment", NTIS AD-A001 260, July 1974.

RICHP74 Richards, P., et al, "Simulation Data Processing Study: Language and Operating System Selection", GE TIS 74CIS09, June 1974.

RICHP75 Richards, P., Chang, P., "Software Development and Implementation Aids IR&D Project Final Report for 1974", GE TIS 75CIS01, July 1975.

RICHP76 Richards, P., Chang, P., "Localization of Variables: A Measure of Complexity", GE TIS 76CIS07, December 1976.

ROSED76 Rosenkrantz, D., "Plan for RDL: A Specification Language Generating System", GE Internal Document, March 1975.

RUBER68 Rubey, R.J., Hartwick, R.D., "Quantitative Measurement of Program Quality", Proceedings of 23rd National Conference, ACM, 1968.

SABIM76 Sabin, M.A., "Portability - Some Experiences with FORTRAN", Software-Practice & Experience, Vol. 6, pp 393-396, 1976.

SACI76 "SAC in Transition", Aviation Week and Space Technology, 10 May 1976.

SACKH67 Sackman, H., Computers, System Science, and Evolving Society, J. Wiley & Sons, 1967.

SALIJ77 Salinger, J., "Initial Report on the Feasibility of Developing a Work Measurement Program for the Data Processing Departments", Blue Cross/ Blue Shield Internal Paper, January 1977.

SALVA75 Salvador, A., Gordon, J., Capstick, C., "Static Profile of Cobol Programs", SIGPLAN Notices, August 1975.

SAMS75 "SAMSO Program Management Plan Computer Program Test and Evaluation", February 1975.

SCHNN72 Schneidewind, N.F., "A Methodology for Software Reliability Prediction and Quality Control", Naval Postgraduate School, NTIS AD-754 377, November 1972.

SCHNN75 Schneidewind, N.F., "Analysis of Error Processes in Computer Software", International Conference on Reliable Software, 1975.

SCHOJ76 Schonfelder, J.L., "The Production of Special Function Routines for a Multi-Machine Library", Software-Practice and Experience, Vol. 6, pp 71-82, 1976.

SCHOW76 Schoeffel, W., "An Air Force Guide to Software Documentation Requirements", NTIS AD-A027 051, June 1976.

SHOOM75a Shooman, M.L., Bolskey, M.I., "Software Errors: Types, Distribution, Test and Correction Times", International Conference on Reliable Software, 1975.

SHOOM75b Shooman, M., "Summary of Technical Progress - Software Modeling Studies", RADC Interim Report, September 1975.

SMITR74 Smith, R.,"Management Data Collection and Reporting - Structured Programming Series (Vol. IX)" RADC TR-74-300, October 1974.

SOFT75 "Software Engineering Handbook", GE Special Purpose Computer Center, September 1975.

SPAC76 "GE Space Division Task Force on Software", Engineering and Management June 28 Report, 1976.

STEWD74 Steward, D.W., "The Analysis of the Structure of Systems", GE TIS 74NED36, June 1974.

SULLJ73 Sullivan, J.E., "Measuring the Complexity of Computer Software", MITRE Tech Report MTR-2648, June 1973.

SUPP73 "Support of Air Force Automatic Data Processing Requirements through the 1980's", SADPR-85, July 1973.

SZABS76 Szabo, S., "A Schema for Producing Reliable Software", International Symposium on Fault Tolerant Computing, Paris, June 1975.

TACT74 "Tactical Digital Systems Documentation Standards", Department of the Navy, SECNAVINST 3560.1, August 1974.

TALIW71 Taliaferro, W.M., "Modularity: The Key to System Growth Potential", Software Practices and Experience, July-September 1971.

TEICD76 Teichroew, D., "PSL/PSA A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems", 1976 Software Engineering Conference.

THAYT76 Thayer, T.A., Hetrick, W.L., Lipow, M., Craig, G.R., "Software Reliability Study", RADC TR-76-238, August 1976.

THAYT75 Thayer, T.A., "Understanding Software through Empirical Reliability Analysis", Proceedings, 1975 National Computer Conference.

USAR75 "US Army Integrated Software Research and Development Program", USACSC, January 1975.

VANDG74 VanderBrug, G.J., "On Structured Programming and Problem-Reduction", NSF TR-291, January 1974 (MF).

VANTD74 Van Tassel, Dennie, Program Style, Design, Efficiency, Debugging and Testing, Prentice-Hall, Inc., New Jersey, 1974.

VOLKW58 Volk, W., Applied Statistics for Engineers, McGraw-Hill Book Co., Inc., New York, 1958.

WALTG74 Walters, G.F., et al, "Spacecraft On-Board Processor/Software Assessment", GE TIS 74CIS10, June 1974.

WALTG76 Walters, G.F., "Software Aids Index", GE Internal Working Paper, December 1976.

WAGOW73 Wagoner, W.L., "The Final Report on a Software Reliability Measurement Study", Aerospace Report TOR-0074, August 1973.

WEING71 Weinberg, G.M., "The Psychology of Computer Programming", NY, Van Nostrand Reinhold, 1971.

WHIPL75 Whipple, L., "AFAL Operational Software Concept Development Program", Briefing given at Software Subpanel, Joint Deputies for Laboratories Committee, 12 February 1975.

WILLN76 Willmouth, N., "Software Data Collection: Problems of Software Data Collection", RADC Interim Report, 1976.

WOLVR72 Wolverton, R.W., Schick, G.J., "Assessment of Software Reliability", TRW Report SS-72-04, September 1972.

WULFW73 Wulf, W.A., "Report of Workshop 3 - Programming Methodology", Proceedings of a Symposium on the High Cost of Software, September 1973.

YOURE75 Yourdon, E., Techniques of Program Structure and Design, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

ZAHNC75 Zahn, C., "Structured Control in Programming Languages", SIGPLAN Notices, July 1975.

BIBLIOGRAPHY

Abernathy, David H., et al.
Survey of Design Goals for Operating Systems
GITIS-72-04, Georgia Institute of Technology, 1972

Discusses the general design goals and their inherent tradeoffs that should be accounted for in the development of Operating Systems. Provides an examination and identification of the design goals of a number of current operating systems. The discussions of the tradeoffs between design goals are relevant to any system development effort.

Boehm, B.W., et al.
Characteristics of Software Quality
TRW 25201-6001-RU-00, 28 Dec 1973

TRW report for National Bureau of Standards. Establishes a definitive hierarchy of related characteristics of software quality and, for each characteristic, defines metrics which, (1) ca be used to provide a quantitative measure for any FORTRAN program, (2) are anomaly-detecting related to the source code, and (3) can be used to define overall software quality. A comprehensive set of examples of good and bad programming practices, related to the anomaly-detecting metrics, is provided. The process of correlating the metrics with the quality characteristics is not explained.

Clapp, J., et al.
MITRE Series on Software Engineering
June 1973

A collection of small studies done by MITRE for ESD. Provides an initial framework for investigation of the problems associated with Software Reliability. Provides some measures of software complexity.

Dennis, J.B., Goos, G., Poole, P.C., Gotlieb, C.C., et al.
Advanced Course of Software Engineering - Lecture Notes
Springer-Verlag, N.Y., 1973

A consolidation of experts' lectures on current topics in Software Engineering. Examples, definitions, and different view points of the solutions to problems are provided.

Drossman, M.M.
Development of a Nested Virtual Machine, Data Structure-Oriented Software Design Methodology and Procedure for Evaluation
USAFOSR/RADC TR, Aug 1976

Provides a brief summary of work done to date in the area of a methodology for software system design. Presents a data-oriented approach to this subject with an example. The approach emphasizes top down successive refinement of the data structure. Outlines a plan for evaluation of the concept.

Fagan, M.
Design and Code Inspections and Process Control in the Development of Programs.
IBM TR 00.2763, June 1976

A manageable, implementable plan for formalizing inspections of documents and code produced during a software development is presented. The concept of inspections is compared with the less formal practice of walk-thrus.

Gilb, T.
Software Metrics
Winthrop Computer Systems Series, 1976

A significant contribution to the field of software quality metrics has been made by this book, even though its presentation detracts from its effect. It provides the most information about the field in one document to date, summarizing previous efforts and introducing some interesting concepts of tests for software qualities.

Goodenough, J.
Effect of Software Structure on Software Reliability, Modifiability, and Reusability, A Case Study.
U.S. Army Armament Command, March 1974

Provides discussion of program structuring concepts and language design features that enhance reusability, modifiability, and reliability. Illustrates concepts by case studies.

Halstead M.
Elements of Software Science
Elsevier Computer Science Library, 1977

This text presents a composite of the work done in the area of software physics or software science. The theory and results of various experiments are provided. Several chapters are devoted to describing applications of the theory.

Kernighan, B., Plauger, P.
The Elements of Programming Style
McGraw-Hill, N.Y., 1974

A comprehensive text on the do's and don'ts of programming. Full of examples which illustrate techniques of expression, structure, input and output, efficiency and instrumentation, and effective documentation. Provides many examples of common faults and errors made during the programming process.

Kosaraju, S.R., Ledgard, H.F.
Concepts in Quality Software Design
National Bureau of Standards Technical Note 842, August 1974

Provides a hierarchy of factors in quality software. The factors represent both measurable qualities and controllable software production characteristics. Brief examples and explanations of the factors are provided and then the areas of prime concern to the authors, top-down programming, proof of correctness, and structured programming are discussed in detail.

Kosy, D.
AF Command and Control Information Processing in the 1980s:
Trends in Software Technology
RAND R-1012-PR, June 1974

A revised and expanded version of the CCIP-85 volume on software. It describes the current (1972) state-of-the-art and forecasts the technology into the 1980's. Identifies quantitative and nonquantitative measures of software quality. Relates many software problems to Air Force requirements.

Lieblein, E.
Computer Software: Problems and Possible Solutions
CENTACS, U.S. Army ECOM Memorandum, November 1972

A discussion of the problem areas in DOD software development and identification of potentially beneficial areas of research and implementation of state-of-the-art techniques as solutions.

Myers, G.J.
Reliable Software Through Composite Design
Petrocelli/Charter, 1975

Describes a set of quality factors for software and identifies which of these factors are influenced by his concept of composite design. Illustrates and defines composite design and provides a classification scheme for looking at software in relationship to how well it is designed.

Myers, G. J.
Software Reliability: Principles and Practices
John Wiley & Sons, N.Y., 1976

A very thorough review of the software reliability state-of-the-art. The book covers reasons why reliability is a major concern in the industry, a working definition of software reliability, principles and practices for designing reliable software, how software should be tested, and briefly a wide range of topics which influence software reliability.

Reifer, D.
Toward Specifying Software Properties
IFIP Working Conference on Modeling of Environmental Systems,
Tokyo, Japan, Apr 76

A concept of specifying software properties and relating them to functional and performance requirements is presented. The presentation of a heuristic approach is very conceptual in nature. The conflicting characteristics of several of the software properties are discussed.

Rubey, R., Hartwick, D.
Quantitative Measurement of Program Quality
Proceedings of ACM National Conference, 1968

One of the initial efforts in the software quality field. Provides a concise mathematical approach to software attributes and corresponding metrics.

Thayer, T.A., et al.
Software Reliability Study
RADC TR-76-238, August 1976

RADC sponsored study to quantify Reliability by correlating source code metrics with the error history of that code. Several AF systems were utilized to perform the correlation. Identifies the metrics, how they can be collected, a categorization of errors, and provides a survey of existing reliability models.

Wulf, W.A.
Programming Methodology - Report of Workshop 3
Proceedings of a Symposium of the High Cost of Software 17-19
September 1973

A tri-service sponsored symposium which identified quality attributes to be used to compare programs. Recommended areas for future research which would add to the state-of-knowledge.

Yourdon, E.
Techniques of Program Structure and Design
Prentice-Hall, Englewood Cliffs, N.J., 1975.

A comprehensive text on top-down structured programming. Discusses the characteristics of a good program, modular programming, and how to program with simplicity and clarity in mind. A good source for the do's and don'ts of programming.

Proceedings of the 1975 International Conference on Reliable Software
IEEE, 1975

Papers by several authors: Dijkstra, Ramamoorthy, Culpepper, Parnas, Edwards, Endres, Manna, Reifer, provide various views on the problems, techniques, and solutions to different aspects of software quality.

Support of Air Force Automatic Data Processing Requirements through the 1980's (SADPR-85)
MITRE, July 1973

A study performed to identify the ADP requirements for AF base level operations through the 1980's. The interesting portion of the study as far as this effort is concerned is Appendix VII which identified the configuration evaluation criteria. A scheme of criteria or qualities with associated priorities were established to compare various proposed configurations for providing the ADP requirements.

APPENDIX A

QUALITY FACTORS REFERENCES IN THE LITERATURE WITH DEFINITIONS

The quality factor definitions or discussions contained in this appendix were found in the literature or generated during this effort to provide additional interpretations. The authors of the quoted or paraphrased definitions are indicated in parentheses.

We wish to emphasize that any errors, omissions, or misleading statements in the quoted or paraphrased definitions contained herein are completely and solely our responsibility and not those of the authors referenced.

PORTABILITY/TRANSFERABILITY

- programs must be readily transferable among different equipment configurations (Goodenough/Culpepper)
- degree of transportability is determined by number, extent, and complexity of changes, and hence the difficulty in implementing a software processor which can mechanically move a program, between a specified set of machines (Hague)
- machine - independence (Marshall)
- measure of the ease of moving a computer program from one computing environment to another (Meahy, Poole)
- how quickly and cheaply the software system can be converted to perform the same functions using different equipment (Kosy)
- an appropriate environment can be provided on most computers (Goos)
- extent that it can be operated easily and well on computer configurations other than its current one (Boehm)
- moving software from one computer (environment) to another (Lieblein)
- measure of the effort required to install a program on another machine, another operating system, or a different configuration of the same machine (Wulf)
- external (black box) form, fit, and function characteristics of a program module which permit its use as a building block in several computer programs (NSSC PATHWAY)
- relates to how quickly and easily a software system can be transferred from one hardware system to another (USA ISRAD)
- ease of conversion of a system from one environment to another; the relative conversion cost for a given conversion method (Gilb)

ACCEPTABILITY

- how closely the ADP system meets true user needs (Kosy)
- measure of how closely the computer program meets the true needs of the user (SAMSO)
- relates to degree to which software meets the user's needs including the clarity and unambiguity of the specifications and the effectiveness of the man-machine interface (USA ISRAD)
- does the software meet the need of the user (Light)

COMPLETENESS

- extent to which software fulfills overall mission satisfaction (McCall)
- extent that all of its parts are present and each of its parts are fully developed (Boehm)

CONSISTENCY

- degree to which software satisfies specifications (McCall)
- extent that it contains uniform notation, terminology, and symbology within itself and the extent that the content is traceable to the requirements (Boehm)

CORRECTNESS

- correctness of its description with respect to the objective of the software as specified by the semantic description of the linguistic level it defines (Dennis)
- the coding of a computer program module which properly and completely implements selected overall system requirements (NSSC PATHWAY)
- relates to degree to which software is free from design and program defects (USA ISRAD)
- the program is logically correct (Rubey)

AVAILABILITY

- fraction of total time during which the system can support critical functions (SADPR-85)
- error recover and protection (Liskov)
- probability that a system is operating satisfactorily at any point in time, when used under stated conditions (Gilb)

RELIABILITY

- includes correctness, testing for errors, and error tolerance (Ledgard)
- the probability that the software will satisfy the stated operational requirements for a specified time interval or a unit application in the operational environment (JLC SRWG)
- the probability that a software system will operate without failure for at least a given period of time when used under stated conditions (Kosy)
- extent to which a program can be expected to perform its intended functions satisfactorily (Thayer)
- ability of the software to perform its functions correctly in spite of failures of computer system components (Dennis)
- probability that a software fault does not occur during a specified time interval (or specified number of software operational cycles) which causes deviation from required output by more than specified tolerances, in a specific environment (Thayer)
- measure of the number of errors encountered in a program (Myers)

RELIABILITY, Continued

- probability that the system will perform satisfactorily for at least a given time interval, when used under stated conditions (Gilb)
- extent to which a program is debugged, can get whatever degree of reliability one is willing to pay for (Casey)
- the probability that the computer program will satisfactorily execute for at least a given period of time when used under specific conditions. (SAMSO)
- tasks are broken into easily manageable modules and programming internal to most modules remains constant (Whipple)
- the degree of assurance that the software will do its job when integrated with the hardware (Light)

ACCURACY

- where mathematically possible a routine should give an approximation that is as close as practicable to the full machine precision for whatever machine it is running on (Schonfelder)
- extent that its outputs are sufficiently precise to satisfy its intended use (Boehm)
- the mathematical calculations are correctly performed (Rubey)
- measure of the quality of freedom from error, degree of exactness possessed by an approximation or measurement (Gilb)

ROBUSTNESS

- routines should be coded so that when it is not possible to return a result with any reasonable accurracy or there is danger of causing some form of machine failure they should detect this and take appropriate actions (Schonfelder)
- quality of a program that determines its ability to continue to perform despite some violation of the assumptions in its specifications (Wulf)
- program should test input for plausability and validity (Kernighan)

EFFICIENCY

- measure of the execution behavior of a program (execution speed, storage speed) (Myers)
- execution time, storage space, # instructions, processing time (Kosy)
- extent that software fulfills its purpose without waste of resources (Boehm)
- the ratio of useful work performed to the total energy expended (Gilb)

EFFICIENCY, Continued

- reduction of overall cost - run time, operation, maintenance (Kernighan)
- extremely fast run time and efficient overlay capabilities (Richards)
- computation time and memory usage are optimized (Rubey)

PERFORMANCE

- the effectiveness with which resources of the host system are utilized toward meeting the objective of the software system (Dennis)
- refers to such attributes as size, speed, precision; e.g., the rate at which the program consumes accountable resources (Wulf)

CONCISENESS

- the ability to satisfy functional requirements with a minimum amount of software (McCall)
- extent that no excessive information is present (Boehm)

UNDERSTANDABILITY

- ease with which the implementation can be understood (Richards)
- reduced complexity, reduced redundancy, clear documentation/notation (Goodenough)
- extent that the purpose of the product is clear to the evaluator (Boehm)
- Documentation remains current (Whipple)
- the program's intelligible (Rubey)

SELF-DESCRIPTIVENESS
CLARITY

- measure of how clear a program is, i.e. how easy it is to read, understand, and use (Ledgard)
- refers to the ease with which the program (and its documentation) can be understood by humans (Wulf)

CLARITY, Continued

- extent that it contains enough information for a reader to determine its objectives, assumptions, constraints, inputs, outputs, components, and status (Boehm)

LEGIBILITY

- extent that its functions and those of its components statements are easily discerned by reading the code (Boehm)

MAINTAINABILITY

- measure of the effort and time required to fix bugs in the program (Myers)
- how easy it is to locate and correct errors found in operational use (Kosy)
- extent that the software facilitates updating to satisfy new requirements (Boehm)
- maintenance involves
  (1) correction to heretofore latent bugs
  (2) enhancements
  (3) expansion
  (4) major redesign (Lieblein)
- ease with which a change can be made due to
  (1) bug during operation
  (2) non-satisfaction of users requirements
  (3) changing requirements
  (4) obsolecence/upgrade of system (McCall)
- probability that a failed system will be restored to operable condition within a specified time (Gilb)

STABILITY

- the "ripple effect" or how many modules have to be changed when you make a change (Myers)
- measure of the lack of percievable change in a system in spite of the occurrence in the environment which would normally be expected to cause a change (Gilb)

ADAPTABILITY

- how much time and effort are required to modify a software system (Kosy)
- measure of the ease with which a program can be altered to fit differing user images and system constraints (Poole)

ADAPTABILITY, Continued

- measure of the ease of extending the product, such as adding new user functions to the product (Myers)
- a measure of the effort required to modify a computer program to add, change or remove a function or use the computer program in a different environment (includes concepts of flexibility and portability) (SAMSO)
- relates to ability of software to operate inspite of unexpected input or external conditions (USA ISRAD)

EXTENSIBILITY

- extent to which system can support extensions of critical functions (SADPR-85)

MODIFIABILITY

- measure of the cost of changing or extending the program (Myers)
- operational experience usually shows the need for incremental software improvements (Goodenough)
- extent that it facilitates the incorporation of changes, once the nature of the desired change has been determined (Boehm)
- quality of a program that reduces the effort required to alter it in order to conform to a modification of its specification (Wulf)
- internal (detailed design) characteristics of a program module are arranged so as to permit easy change (NSSC PATHWAY)
- use of HOL reduces programmer's task and human errors and allows smaller units to be tested permitting easier de-bugging (Whipple)
- the program is easy to modify (Rubey)

ACCESSIBILITY

- extent that software facilitates the selective use of its components (Boehm)

FLEXIBILITY
EXPANDABILITY
AUGMENTABILITY

- extent to which system can absorb workload increases and decreases which require modification (SADPR-85)
- the ability of a system to immediately handle different logical situations (Gilb)

FLEXIBILITY
EXPANDABILITY
AUGMENTABILITY, Continued

- how easily the software modules comprising the system or subsystem can be rearranged to change the system's functions without modifying any of the modules (Kosy)
- ease of changing, expanding, or upgrading a program (Yourdon)
- the software modules must be usable in a variety of contexts (Culpepper)
- includes changeability; e.g., the ease of correcting bugs, maintenance because of changing specifications, and portability to move to another system (Ledgard)
- extent that software easily accommodates expansions in data storage requirements or component computational functions (Boehm)
- attributes of software which allow quick response to changes in algorithms (Richards)
- ability to reuse the software and transfer it to another processor (includes reuse, adaptability, transferability and versatility of software) (Light)

INTEGRITY

- how much the operation of one software subsystem can protect the operation of another (Kosy)
- a measure of the degree of protection the computer program offers against unauthorized access and loss due to controllable events (includes the concepts of privacy and security) (SAMSO)
- relates to ability of software to prevent purposeful or accidental damage to the data or software (USA ISRAD)
- ability to resist faults from personnel, the security problem, or from the environment, a fault tolerance issue (Light)
- probability of system survival when subjected to an attack during a time interval (Gilb)

SECURITY

- the ability to prevent unauthorized access to programs or data (Kosy)
- extent to which access to software, data, and facilities can be controlled (SADPR-85)
- measure of the probability that one system user can accidentally or intentionally reference or destroy data that is the property of another user or interface with the operation of the system (Myers)

SECURITY, Continued

- relates to the ability of the software to prevent unauthorized access to the system or system elements (USA ISRAD)

PRIVACY

- the extent to which access to sensitive data by unauthorized persons can be controlled and the extent to which the use of the data once accessed can be controlled (McCall)
- relates to the protection level for data in the system and the individual's right to review and control dissemination of data (USA ISRAD)

USABILITY
OPERABILITY

- measure of the human interface of the program (Myers)
- ease of operation from human viewpoint, covering both human engineering and ease of transition from current operation (SADPR-85)
- how suitable is it to the use (Kosy)
- software must be adequately documented so that it can be easily used and maintained (Culpepper)
- extent that it is convenient and practicable to use (Boehm)
- the program is easy to learn and use (Rubey)

HUMAN FACTORS

- every program presents an interface to its human users/operators, by human factors we refer collectively to all the attributes that make this interface more palatable: ease of use, error protectedness, quality of documentation, uniform syntax, etc. (Wulf)
- extent that software fulfills its purpose without wasting user's time and energy or degrading their morale (Boehm)
- measure of the product's ease of understanding, ease of use, difficulty of misuse, and frequency of user's errors (Myers)

COMMUNICATIVENESS

- extent that software facilitates the specifications of inputs and provide outputs whose form and content are easy to assimilate and useful (Boehm)

STRUCTUREDNESS
MODULARITY

- ability to combine arbitrary program modules into larger modules without knowledge of the construction of the modules (Goos)
- the software must consist of modules with well defined interfaces. Interactions between modules must occur only at those interfaces (Culpepper)
- extent that it possesses a definite pattern of organization of its independent parts (Boehm)
- how well a program is organized around its data representation and flow of control (Kernighan)
- there is no interference between program entities (Rubey)
- formal way of dividing a program into a number of sub units each having a well defined function and relationship to the rest of the program (Mealy)

UNIFORMITY

- module should be usable uniformly (Goodenough)

GENERALITY
REUSABILITY

- measure of the scope of the functions that a program performs (Myers)
- building programs from reusable software modules can significantly reduce production costs (Goodenough)
- how broad a class of similar functions the system can perform (Kosy)
- standardized modules can be lifted from one program and used in another without extensive re-coding or re-testings (Whipple)
- degree to which a system is applicable in different environments (Gilb)

TESTABILITY

- instrumentation and debugging aids (Liskov)
- minimize testing costs (Yourdon)
- provision of facilities in the design of programs which are essential to testing complex structures (Edwards)
- extent that software facilitates the establishment of acceptance criteria and supports evaluation of its performance (Boehm)
- a measure of the effort required to exercise the computer program to see how well it performs in a given environment and if it activally solves the problem it was supposed to solve (SAMSO)

TESTABILITY, Continued

- measure of our ability to test software (Light)

INTEROPERABILITY

- how quickly and easily one software system can be coupled to another (Kosy)
- relates to how quickly and easily one software system can be coupled to another (USA ISRAD)

CONVERTIBILITY

- degree of success anticipated in readying people, machines, and procedures to support the system (SADPR-85)

MANAGEABILITY

- degree to which system lends itself to efficient administration of its components (SADPR-85)

COST

- includes not only development cost, but also the costs of maintenance, training, documentation, etc., on the entire life cycle of the program (Wulf)
- there are three major categories of cost:

  Economy of operation - relates to cost or operating system

  Economy of modification - relates to cost of making changes to software to meet new requirements or correct defects resulting in errors in requirements, design, and programming

  Economy of development - relates to cost of entire development cycle from identification of requirement to initial operation
- development and maintenance costs (Myers)
- implementation cost and operational cost (Gilb)

ACCOUNTABILITY

- extent that code usage can be measured (Boehm)

SELF-CONTAINEDNESS

- extent to which a program performs all its explicit and implicit functions within itself (Boehm)

EXPRESSION

- how a program is expressed determines in large measure the intelligibility of the whole (Kernighan)

VALIDITY

- relates to degree to which software implements the user's specifications (USA ISRAD)

TIME

- two major categories of time:

  Modification Time - relates to total elapse time from point when new requirement or modification is identified the change is implemented and validated

  Development Time - relates to total elapsed time of development (USA ISRAD)

- development time (Myers)
- what is the expected life span of system (Gilb)

COMPLEXITY

- relates to data set relationships, data structures, central flow, and the algorithm being implemented (Richards)
- measure of the degree of decision-making logic within a system (Gilb)

DOCUMENTATION

- quality and quantity of user publications which provide ease of understanding or use (Myers)

PRECISION

- the degree to which calculated results reflect theoretical values (Gilb)

TOLERANCE

- measure of the systems ability to accept different forms of the same information as valid or withstand a degree of variation in input without malfunction or rejection (Gilb)

COMPATABILITY

- measure of portability that can be expected of systems when they are moved from one given environment to another (Gilb)

REPAIRABILITY

- probability that a failed system will be restored to operable condition within a specified active repair time when maintenance is done under specified conditions (Gilb)

SERVICEABILITY

- degree of ease or difficulty with which a system can be repaired (Gilb)

APPENDIX B

DOCUMENTATION CHARACTERISTICS

SOFTWARE SYSTEM REQUIREMENTS SPECIFICATION AND REVIEW

This document describes the functional requirements and capabilities of the software system. The data requirements, essentially from the view point of the end user, are described. It includes the operational constraints and considerations imposed on the software by the hardware system. It constitutes the primary interface between the user/customer and the developer, during both formal and informal reviews. As such, it is a primary reference during the design and development of a software system. Because of this, any revisions are continually made to keep it as current as possible.

STANDARDS AND CONVENTIONS

This document provides guidelines for the design and implementation of computer programs. A standard is defined to be a rule which must be followed to produce an acceptable product. A convention is defined to be a recommended procedure which will enhance the quality of a program's operation or its documentation. Many of the checklist type metrics described are related to standards and conventions. The implications of standards and conventions go beyond the production of correct and reliable software in their goal of achieving a consistent, standardized product which will be easier to maintain and understood by personnel several years after delivery.

DOCUMENTATION PLAN

The Documentation Plan defines the purpose, scope, usage, content and format of all deliverable documentation. When used in conjunction with the associate contractor's Contract Data Requirements List (CDRL), it provides direction for the preparation of CDRL items as well as establishing acceptance criteria for appropriate documents. This document, by virtue of its being referenced in the SOW and CDRL, becomes an integral part of the contract. Its contents are:

Chapter I --- Descriptions of documents which explain the technical aspects of software requirements, design and development, and computer listings of the approved software routines and data structure and data.

Chapter II -- Describes those documents which relate to the overall management of the contract.

Chapter III - Describes those documents which enable the contractor to report items of interest to other agencies and organizations as well as to the customer.

Chapter IV -- Describes those documents which deal directly with the management, execution and results of the contractor's official testing program.

Chapter V --- Presents descriptions of all documents and forms which allow the contractor to exercise proper configuration control of all the software, data and documentation.

Appendix A -- Presents a discussion of document issuing, updating and revision procedures.

Appendix B -- Depicts the format of the computer usage forecast and utilization report.

Appendix C -- Depicts all of the configuration control forms.

MANAGEMENT PLAN

This document describes the managerial approach and procedures that will be followed by the contractor to perform the contract tasks. It covers the entire spectrum of activities from development, validation, to operation and maintenance.

PRELIMINARY DESIGN SPECIFICATION

This document presents the implementation concepts at the subsystem level. The performance characteristics of the software system are considered. The components of the system described are as follows:

Scope
Overview of the System
Function Design
Data Base Design
Operational Design
RFQ Compliance

PRELIMINARY DESIGN REVIEW

Performance characteristics, changes to system specification, changes to preliminary design, testing plans, and interface requirements are discussed and approved at PDR. All of these items are documented for future reference and required changes identified as action items for resolution.

DETAILED DESIGN SPECIFICATIONS (Parts I and II)

The detailed design specification describes the design at the beginning of the coding (build to - Part I) phase and at the end (built to - Part II). The Part I specifications provides a subsystem description which illustrates the design and operating concepts. Typically it covers:

- Subsystem Description
- Requirements Satisfaction
- Design Concepts
- Operating Concepts
- Function Description
- Subsystem Input/Output
- Subsystem Storage and Timing
- Subsystem Limitations

It also presents a detailed description of each function in the subsystem as follows:

- Purpose
- Description
- Usage
- Inputs
- Processing
- Outputs
- Storage, Timing and Restrictions

Part II is an updated version of the Part I specifications plus the actual listings of the functions implementation.

CRITICAL DESIGN REVIEW

Involves review and approval of the Detailed Design (Part I), the test plan procedures, and any problem reports and their resolutions. The problem reports may involve requirements, design, and coding problems.

VALIDATION AND ACCEPTANCE TEST SPECIFICATION

Validation and acceptance testing is directed at the verification of satisfaction of the contract baseline requirements. It provides the testing strategy and design to provide a validation of the functional operation of the software. A typical outline is as follows:

- Introduction
- Purpose
- Scope
- Reference
- Testing Structure
- Test Program Controls
- Development Testing Summary
- V & A Testing
- Test Support

USER'S MANUAL/OPERATOR'S MANUAL

These documents provide the user/operator all of the information required to use the software and operate the system. They include descriptive data about the Data Base, deck setups, commands, inputs, outputs, error messages, recovery techniques, training and instructional information sources, and maintenance. The documents are handbook oriented and become an operational tool during system operation.

INTERFACE CONTROL DOCUMENT

The purpose of this document is to identify all system, subsystem, and function-level interfaces and describe all pertinent data associated with them. It formally specifies them for review at CDR and for implementation.

CONFIGURATION MANAGEMENT PLAN

The primary objective of this plan is to establish and implement a formal set of uniform procedures which will provide each subsystem developer with the maximum latitude for the independent management of the software configuration for which they are contractually responsible, yet, imposes the necessary degree and depth of control for ensuring that the identity and integrity of the entire software system is maintained. The document describes the multiple configuration management baselines, the critical events during the development timeline, and the configuration controls and procedures employed during software development and testing.

DATA BASE MANAGEMENT PLAN

This plan describes the roles, responsibilities and schedules necessary to insure accurate, complete and timely preparation and delivery of COMPOOLS, Data Bases and Data Base User's Guides to support the software system from initial design through operation. Included in this effort is the development of the Data Definition Specification which describes the data interfaces and structures at the element level and the Data Base User's Guide which provides the user with descriptive information required to use and change the Data Base.

PROBLEM IDENTIFICATION AND CORRECTION REPORTS

Design Problem Report (DPR)

The DPR is the primary means of documenting problems in the critical design review material and transmitting that information to the appropriate contractor for corrections. DPR's will be superseded by the use of the SPR upon approval of the proposed design documentation by the customer.

Software Problem Report

The SPR will be utilized:

A. To report suspected problems in an existing software routine, Auxiliary COMPOOL, data base, and/or approved baseline documentation.

B. At the option of the customer, to notify developers of new or proposed design requirements and to authorize the initiation of preliminary design review materials.

C. To report suspected problems in the system software and associated documentation.

Document Update Transmittal (DUT)

The DUT is the primary means by which proposed changes to existing documentation are transmitted for review and approval of the customer. A DUT package consists of the DUT form and change pages in preliminary form, and is normally initiated in response to a DPR or SPR.

COMPOOL Change Request (CCR)

The CCR is utilized to request, coordinate, and control all changes to the official COMPOOL. The CCR will be written in response to an SPR that requires a COMPOOL change. (The CCR will be referenced on the MTM initiated in response to the SPR.

Data Base Change Request (DBCR)

The DBCR is utilized to request, coordinate, and control all changes to the official data base. It is the only method of coordinating data base changes between the users of the data base and the agency responsible for implementation and maintenance. The DBCR is also utilized to close out an SPR that requires a data base change. (The DBCR will be referenced on the MTM initiated in response to the SPR.)

Modification Transmittal Memorandum (MTM)

The MTM is utilized for the following:

A. To supply an explanation intended to close an SPR where no corrections or revisions are required.

B. To transmit changes for incorporation into existing programs.

C. To transmit a new or modified routine for inclusion on an official tape.

D. To indicate a need for revisions to documentation.

E. An explanation indicating that an accompanying COMPOOL change or data base change is to close out an SPR.

TRAINING MATERIAL

Training can include lesson plans, exercises, and courses of instruction covering both user and operator aspects of the system.

PHASES	INITIATION	DEVELOPMENT			EVALUATION	OPERATION
		DEFINITION	DESIGN	PROGRAMMING	INTEGRATION, SYSTEM TEST, INSTALLATION	OPERATION, MAINTENANCE
FUNCTIONS PERFORMED	SYSTEM REQUIREMENTS SPECIFICATION	SOFTWARE SYSTEM REQUIREMENTS ANALYSIS	PRELIMINARY DESIGN / DETAILED DESIGN	CODE & DEBUG / DEVELOPEMENT TESTING	SUBSYSTEM TESTING / INTEGRATION / SYSTEM TESTING	OPERATION / REVISION / TRANSISTION
DOCUMENTATION	• SYSTEM REQUIREMENTS SPECIFICATIONS	• SOFTWARE SYSTEM SPECIFICATIONS • MANAGEMENT PLAN • STANDARDS & CONVENTIONS • DOCUMENTATION PLAN • DATA BASE AM • DATA BASE MANAGEMENT PLAN • TEST PLAN MANAGEMENT	• PRELIMINARY DESIGN SPEC (PART I) • PRELIMINARY DATA BASE REQUIREMENTS SPEC • INTERFACE CONTROL DOCUMENT • OPERATOR INTERFACE DOCUMENT • DATA BASE REQUIREMENTS SPEC • DETAILED DESIGN SPC PART II (BUILT TO) • TEST REQUIREMENTS	• SOURCE CODE • VALIDATION & ACCEPTANCE TEST SPEC (TEST PLANS & PROCEDURES) • DETAILED DESIGN SPEC PART II (BUILT TO) • INTEGRATED TEST PLAN • SUBSYSTEM TEST REPORT • INTEGRATED TEST REPORT • USERS MANUAL	
BASELINES		△ SOFTWARE REQUIREMENTS BASELINE	△ DETAILED DESIGN BASELINE	△ SOFTWARE PRODUCT BASELINE		△ OPERATIONAL CONFIGURATION BASELINE
CONFIGURATION CONTROL DOCUMENTS			• DESIGN PROBLEM REPORTS • SOFTWARE PROBLEM REPORTS • SOFTWARE ANALYSIS REPORTS • COMPOOL CHANGE REQUEST • DATA BASE CHANGE REQUEST		• DOCUMENT UPDATE • MODIFICATION TRANSMITTAL MEMO	
REVIEWS		△ SOFTWARE REQUIREMENTS REVIEW	△ PRELIMINARY DESIGN REVIEW (PDR) △ CRITICAL DESIGN REVIEW	△ VALIDATION & ACCEPTANCE TEST PLAN REVIEW	△ INTEGRATED TEST PLAN REVIEW	△ TURNOVER REVIEW

16501-1

Figure B-1 Software Products

Table B-1 Cross Reference Between Identified Documents and References Where They are Described

	BOEHB73	CONF64	CONF66	COMP66a	COMP66b	COMP69	DOCU75	DODM72	HAGAS75	MILI68	MILI70	PILIM68	SCHOW76	TACT74
Software System Requirements Specification	●	●		●	●	●	●	●		●	●	●	●	●
Software Requirements Review	●			●	●	●			●					
Standards and Conventions					●									
Documentation Plan													●	
Management Plan					●								●	
Preliminary Design Specification	●			●	●	●	●	●		●	●	●	●	●
Preliminary Design Review		●	●	●	●	●			●			●		
Detailed Design Specifications	●		●	●	●	●	●	●		●	●	●	●	●
Critical Design Review		●	●	●	●	●			●			●		
Validation and Acceptance Test Specification and Review	●		●	●	●	●	●	●	●			●	●	●
User's/Operator's Manual	●			●	●		●	●				●	●	●
Interface Control Document				●	●	●					●		●	●
Configuration Management Plan					●							●	●	
Data Base Management Plan and User's Guide							●	●					●	●
Problem Reports						●							●	
Training Material													●	

1670

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